UNIVERSIDAD PONTIFICIA COMILLAS
ESCUELA TECNICA SUPERIOR
DE INGENIEROS INDUSTRIALES
I.C.A.I.

PROYECTO DE DESARROLLO E IMPLANTACION DE ANA: HERRAMIENTA PARA EL DISEÑO Y ESPECIFICACION DE ALGORITMOS

MADRID, JUNIO DE 1992
JUAN RIVIER ABBAD


Técnicarp

Universidad Pontificia Comillas
Escuela Técnica Superior de Ingenieros Industriales
I.C.A.I.

Proyecto de

Desarrollo e implantación de ANA: Herramienta para el diseño y especificación de algoritmos

Madrid, a 2 de Junio de 1992

Juan RIVIER ABBAD

DOCUMENTO N°1

MEMORIA

MEMORIA

Índice

Capítulo 1

INTRODUCCIÓN

Actualmente, disciplinas tan variadas como la Ingeniería del Software, la Inteligencia Artificial, los Sistemas Expertos, etc. se dedican básicamente a una tarea fundamental: conseguir que las máquinas realicen o emulen procesos de razonamiento. Este conjunto de actividades se considera aquí como distintos enfoques de una disciplina común llamada la Ingeniería del Conocimiento.

Un proyecto de Ingeniería del Conocimiento se enfrenta a grandes dificultades:

- Acceso al conocimiento específico del problema: el programador, o ingeniero del conocimiento, tiene unos conocimientos generales (física, matemáticas, etc.) pero estos no suelen ser suficientes para poder realizar un programa sobre un tema específico. Esto implica que debe documentarse de manera eficaz y trabajar en colaboración con un experto en la materia a tratar.

- Acceso a las necesidades del cliente (o definición de requisitos del usuario): estas necesidades no siempre están muy claras. Debe existir un continuo contacto entre ambos para garantizar que se va por buen camino.

- Realización del proyecto bajo unas restricciones importantes de tiempo y presupuesto: el diseño del programa debe ser eficaz.

- Necesidad de especificación exhaustiva del producto realizado: debe poder ser comprensible, para el caso en que el cliente quiera utilizarlo para futuras versiones o simplemente mejorarlo.

Para resolver estas dificultades, se propone aquí una metodología sistemática basada en la representación del conocimiento mediante un lenguaje gráfico de programación (lenguaje y metodología ANA).

Al ser los lenguajes gráficos de muy alto nivel (gran eficacia en la transmisión de información), pueden ser comprensibles por el experto en la materia o el cliente, que pueden participar en la elaboración del algoritmo aportando conocimientos o requisitos que debe cumplir. Además el lenguaje nos permite diseñar el programa con suficiente detalle como para que la implantación y el depurado de errores se simplifiquen al máximo. Y por último nos deja una especificación del programa fácilmente comprensible por cualquiera. De aquí surge la necesidad de una herramienta informática que nos permita manejar este lenguaje con cierta comodidad.

Este es el principal objetivo de este proyecto: diseñar e implantar la herramienta informática ANA. Este tipo de herramientas se denominan en la industria del software como herramientas CASE (Computer Aided Software Engineering).

Se pretende entonces desarrollar una herramienta informática capaz de representar automáticamente los dibujos propios del lenguaje gráfico ANA, partiendo de instrucciones de alto nivel del usuario. Esto quiere decir que el usuario de la herramienta sólo necesite especificar relaciones abstractas entre funciones y variables, encargándose la propia herramienta de traducirlas unívocamente a su representación gráfica.

La herramienta debe funcionar en un entorno PC-compatible (lo que implica fuertes restricciones de tiempo y memoria) y presentar unas prestaciones gráficas de nivel profesional: es decir menús, ventanas, utilidades, capacidad de impresión de alta calidad, etc. .

Con estos requisitos se desarrolló la primera versión de la herramienta. Sus mayores limitaciones son la incapacidad de representar un proyecto global en su conjunto, limitándose a la representación del comportamiento interno de una función (diagramas ANA).

La impresión de estos diagramas se realiza por medio de ficheros escritos en lenguaje PostScipt. El código esta escrito en C, intentando así aprovechar al máximo los recursos de la máquina.

Esta primera versión funciona actualmente y se ha empleado con éxito en una veintena de proyectos comerciales en el Instituto de Investigación Tecnológica (Universidad Pontificia Comillas). Con objeto de ampliar las prestaciones de la herramienta, está actualmente en

desarrollo la segunda versión de la misma, cuyas principales mejoras respecto de la primera son:

- Gestión global de todo un proyecto.

- Representación de grandes árboles de variables y de funciones (además de los diagramas ANA).

- Gestión simultánea de un grupo de diagramas ANA (integración o partición de unos diagramas en otros).

- Configuración personalizada para cada usuario.

- Implantación en lenguaje Standard C pensando en su transporte a otros entornos de hardware y otros sistemas operativos distintos del DOS, e importantes mejoras en la gestión de memoria.

- Gestión de ficheros en distintos directorios.

El diseño de esta segunda versión esta prácticamente acabado pero faltan las etapas de implantación y depurado.

En el futuro se pretende incorporar a la herramienta la generación de pseudo-código en lenguaje C a partir de un Estándar ANA.

Capítulo 2

OBJETIVO DEL PROYECTO

2.1- Presentación

El objetivo de este proyecto es conseguir una herramienta informática de ayuda al programador tanto a alto nivel (diseño global) como a bajo nivel (generación de código). En estos momentos queremos incidir en los procesos de diseño del software y su especificación. Esta ayuda no es en ningún caso trivial. Repasemos un poco los problemas que suelen aparecer durante un proyecto de software.

En un principio el programador suele interrogar repetidamente al cliente. Después de esto hay un proceso de creación del software al final del cual aparece un programa que se supone era lo que el cliente deseaba: concretamente su creador espera satisfacer las necesidades del cliente con ese producto.

Este proceso tiene unas características muy definidas: ausencia de líneas de actuación definidas, falta de estandarización, documentación mínima y un sin fin de etc. . Una vez acabado es imposible retocarlo y adaptarlo a las verdaderas necesidades del usuario. Y sobre todo es

impensable que otro que no sea su creador intente comprenderlo partiendo del código.

Poco a poco se han ido desarrollando técnicas de programación para intentar resolver estos problemas: apareció la ingeniería del software.

La característica esencial del software común a otros productos de ingeniería es su complejidad. Es imposible intentar resolver un problema importante de forma centralizada y global. Hay que enfrentarse a problemas específicos y bien definidos. Para ello se reduce la complejidad de un problema en subproblemas y así sucesivamente hasta que estos sean asequibles a la mente humana. Para conseguir esto de forma ordenada aparecieron las herramientas CASE (Computer Aided Software Engineering). Estas pretenden resolver los problemas que presenta la creación de programas complejos mediante la ayuda de gráficos. Es equiparable a los planos que se utilizan en las otras ramas de la ingeniería o en arquitectura. Este proyecto intenta colaborar en ese esfuerzo.

2.2- Objetivos

Lo que se intenta en este proyecto es conseguir una herramienta con la cual sea más fácil trabajar en un proyecto de software.

Lo primero que intenta ser esta herramienta es una ayuda al diseño de algoritmos en general. Utilizando un lenguaje gráfico fácil de comprender llegamos a diseñar un algoritmo en su totalidad.

Para ello definimos el problema en varios niveles de complejidad. Partimos del más alto que describimos con grandes bloques de actuación para luego pasar a describir éstos, y así sucesivamente. Conseguimos aquí pues el objetivo primordial de todo programador que es tratar con problemas de una complejidad suficientemente limitada como para poder resolverlos eficientemente. De esta manera aseguramos que el programa sea totalmente modular y con sus módulos perfectamente definidos.

El segundo punto que quiere resolver este proyecto es el de la especificación del programa realizado. La ventaja que representa esto es que tenemos todo el producto descrito en un lenguaje gráfico y por tanto de muy alto nivel, comprensible para todo el mundo. Ya no hace falta conocer el lenguaje de programación para poder entender como esta hecho el programa. Y por supuesto el mantenimiento es infinitamente más fácil. Solo tenemos que meternos en el nivel de complejidad que haga falta y modificar únicamente módulos perfectamente definidos por la especificación. En otras palabras, nos permite manejar el menos código posible a la vez. Además tenemos la ventaja que si hemos diseñado el producto con este lenguaje ya tenemos hecha la especificación (al ser esta lo último que se suele hacer en un proyecto, no

se suele invertir demasiadas ganas ni tiempo en su realización y por consecuente el resultado suele ser bastante pobre).

El tercer punto es una consecuencia de los dos anteriores. Durante el desarrollo de un programa uno de los problemas principales es el contacto permanente entre todos los implicados en el proyecto: desde el jefe del proyecto hasta el cliente pasando por los distintos programadores que estén trabajando en él. El utilizar este lenguaje para el diseño permite una fácil distribución del trabajo a realizar al estar todas las partes del programa totalmente definidas: tenemos los datos de entrada, lo que hay que hacer, y que datos hay que obtener.

Al mismo tiempo, el cliente puede participar en el desarrollo del proyecto o sencillamente conocer como va, ya que tiene un plano del mismo. Esto le permite definir mucho más sus deseos y por lo tanto llegar a un producto que le satisfaga plenamente. Esto último es muy importante.

De todos estos objetivos sacamos unos requisitos para la herramienta informática a desarrollar.

2.3- Requisitos

Lo primero que hay que mencionar es que el lenguaje gráfico seleccionado para esta metodología es el lenguaje ANA (ver Capítulo 5).

Se puede comprobar que este apartado está dividido en la descripción de los requisitos de dos versiones distintas. Esti es porque una vez acabada la primera versión se vió la necesidad de realizar una segunda con importantes mejoras. Cada una presenta sus propias características.

2.3.1- Versión 1

* Esta herramienta va a ser utilizada para crear diagramas ANA. Por lo tanto debe estar en un entorno gráfico.

* El interfaz con el usuario tiene que ser de alto nivel y tutorial: solo permite diagramas formalmente correctos. Es decir que no se permitirá hacer cosas imposibles en un lenguaje de programación estructurada.

* La estética se cuidará al máximo:

  - Habrá colores para todo de manera que se puedan cambiar.

  - Las ventanas de los menús serán "profesionales" (i.e. con opciones accesibles por cursor o letra clave, posibilidad de 'AvPag' y RePag' en caso de tener demasiadas opciones, video inverso en la opción elegida, etc.).

- En el diagrama se intentará evitar los cruces de líneas innecesarios y aprovechar al máximo el espacio disponible.

* Se deben poder realizar diagramas que ocupen más de una pantalla y realizar por tanto todo el interfaz necesario para poder manejarlos. También se podrán ver en distintas escalas.

* Los diagramas se almacenarán en ficheros. Estos se podrán llamar desde la herramienta para poder trabajar con ellos en distintas sesiones.

* La herramienta correrá en un ordenador personal PC compatible para su fácil instalación y difusión, con las consiguientes limitaciones de memoria y velocidad de proceso.

* Se deben poder imprimir los diagramas ANA en impresoras de alta calidad (LASER) mediante un lenguaje gráfico estándar, como por ejemplo el PostScript.

2.3.2- Versión 2

Esta versión debe cumplir todos los requisitos de la anterior más otros que pasamos a describir:

* El código debe estar preparado para hacer versiones en otros entornos que no sean PC-compatibles. Para ello se utilizará sólo

Standard C, se aislará lo referente al sistema operativo y lo referente a los gráficos (de esta manera solo habrá que reemplazar esos módulos).

* En esta versión aparecerán otros entornos en los cuales se podrán describir los árboles de funciones y de variables.

* Se debe poder verificar la consistencia entre los diversos diagramas ANA. No hay que olvidar que cada uno de ellos es una función que esta relacionada con las otras a través de diagramas ANA y del árbol de funciones.

* Esta versión tendrá un traductor para importar ficheros *.ANA de la versión anterior.

* Habrá una configuración accesible de la herramienta más completa que en la versión anterior y ésta se podrá guardar en un fichero de manera a poder recuperar esa misma configuración en posteriores sesiones de trabajo.

Capítulo 3

INGENIERÍA DEL CONOCIMIENTO

3.1- Introducción

La inteligencia artificial, la programación matemática o la ingeniería de software son distintas maneras de acercarnos al mismo problema: que las máquinas resuelvan situaciones que necesitan procesos de razonamiento. Podemos por tanto decir que forman parte de una misma rama de investigación: la Ingeniería del Conocimiento.

En este proyecto se intenta mostrar una manera de aproximarse a este problema que ya ha sido aplicada con éxito en la práctica a proyectos reales. Es una línea de trabajo orientada a la máquina para que las posibilidades de esta se utilicen al máximo. Siempre que los ordenadores se basen en procesadores y memorias, estarán destinadas a seguir un modelo función/variable de resolución de problemas. Todo lo que no sea esto deberá ser simulado.

Se puede definir esta rama de la Ingeniería del Conocimiento como racionalista, matemática, procedural y basado en métodos de representación gráficos. Racionalista, porque se basa en modelos abstractos de la realidad. Matemático, porque usa modelos matemáticos de esa misma realidad (solo utiliza el modelo función/variable).

Procedural ya que el elemento clave en la resolución del problema es la función (o proceso). Y por último basado en métodos de representación gráficos ya que un proyecto de Ingeniería del Conocimiento consiste en construir un sistema complejo de funciones y variables que como todo sistema complejo en cualquier campo de la ingeniería necesita planos.

Resolver un problema consta de dos etapas básicas: definir qué hay que hacer y definir cómo hacerlo. El 'qué hay que hacer' es una descripción externa del problema mientras que el 'cómo hacerlo' es una descripción interna. Esta descripción interna comprende una serie de subproblemas relacionados entre si según un cierto algoritmo. El descomponer cada problema en una serie de subproblemas es el camino de crear un sistema complejo de funciones y variables.

3.2- Desarrollo sistemático de un proyecto de Ingeniería del Conocimiento

En las carreras técnicas en general los estudiantes son evaluados fundamentalmente por su capacidad de aplicar conocimientos teóricos al modelado de situaciones nuevas, es decir, formalizar y resolver numéricamente problemas enunciados textualmente. Un proyecto de Ingeniería del Conocimiento no es más que un problema, planteado a profesionales con experiencia, para la resolución del cual se dispone

típicamente de varios meses y de toda la documentación que se considere necesaria.

Una de las dificultades clave en estos proyectos es obtener el conocimiento de aquellas personas expertas en el problema que se desea resolver. Dichas personas son capaces de responder a preguntas muy concretas sobre sus conocimientos, pero a veces no es fácil plantear ni tampoco responder preguntas generales que proporcionen una estructura de base al problema.

La segunda gran dificultad consiste en realizar un modelo adecuado del problema y de su proceso de resolución. Cuando el problema es suficientemente complejo, el modelado puede dar lugar a un número muy grande de subproblemas y de datos interrelacionados que hay que mantener siempre bajo control.

Por último, la implantación informática del sistema destinado a resolver el problema puede ser una tarea muy complicada, tanto en su planificación y desarrollo como en la validación sistemática y búsqueda de errores.

A lo largo de este apartado se verá cómo estas tres grandes tareas se pueden afrontar con más garantías de éxito siguiendo un método sistemático que se apoya, fundamentalmente, en un buen sistema de representación gráfica del conocimiento.

3.2.1- Adquisición del conocimiento y modelado de un problema

El desarrollo sistemático de un proyecto de Ingeniería del Conocimiento comprende tres etapas: adquisición del conocimiento, modelado e implantación. El proceso conjunto está representado en forma de algoritmo en la figura -3.1-. Todos los procesos tienen en común el generar una representación formal de sus resultados (llamada "Represent." en la figura -3.1-).

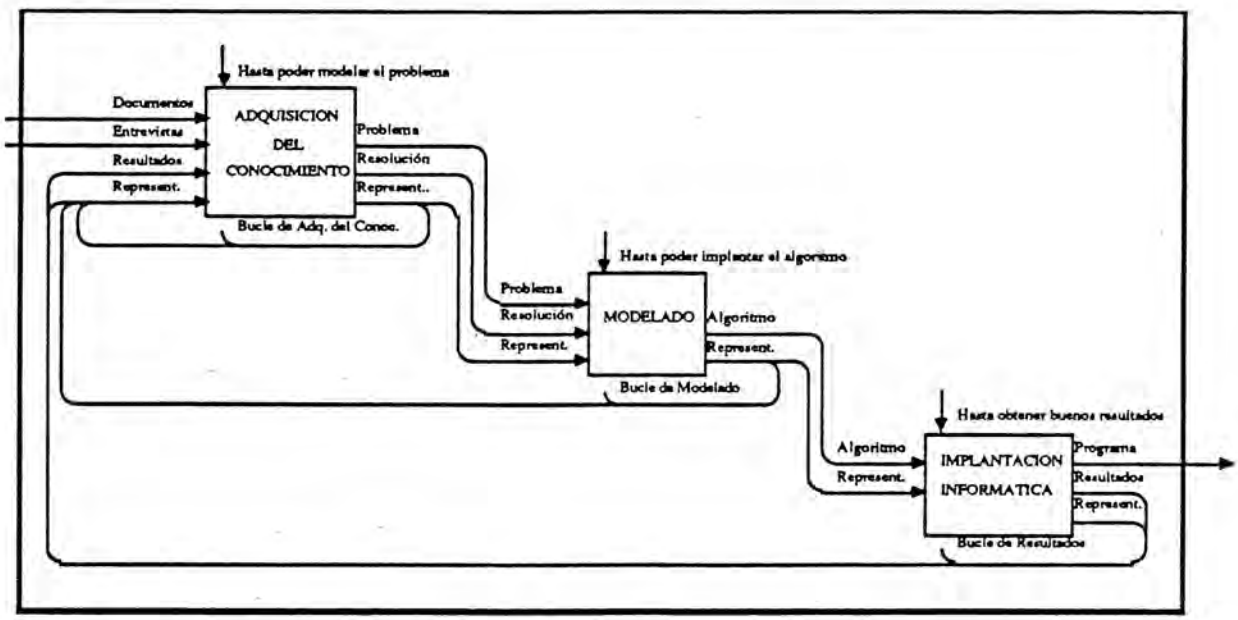


figura -3.1-

Desarrollo sistemático de un proyecto de Ingeniería del Conocimiento.

Un sistema automático de resolución de un problema consiste en la implantación informática de un algoritmo. A su vez, el desarrollo de un algoritmo requiere la adquisición del conocimiento necesario y el modelado tanto del problema en sí como de su resolución.

El desarrollo de un sistema automático de resolución de un problema requiere un conocimiento profundo y preciso del problema que se desea resolver. Es muy frecuente que las personas que poseen ese conocimiento carezcan del tiempo y de la objetividad necesarios para formalizar el problema exhaustivamente. Es aquí donde interviene el ingeniero del conocimiento, que deberá contar con una base suficiente en disciplinas generales (Matemáticas, Física, Economía, etc..) y una fuerte base en Lógica aplicada, a ser posible unidas a la experiencia y la creatividad.

La adquisición del conocimiento sobre las variables y los procesos que comprende la resolución de un problema es una tarea llevada a cabo por el ingeniero del conocimiento (abreviado IC) en colaboración con el experto en el problema. Esta tarea se puede facilitar enormemente con la ayuda de un método gráfico de representación, siguiendo el siguiente método sistemático de aprendizaje y representación formal:

i) El experto proporciona documentación general y ejemplos sobre el problema al IC; el IC proporciona al experto documentación general y ejemplos del método gráfico de representación usado.

ii) El IC revisa la documentación y trata de modelar el problema, representándolo según el método gráfico.El modelado será en general incompleto y revelará lagunas de información y defectos de comprensión en el conocimiento manejado por el IC.

iii) El experto proporcionará documentación adicional y aclarará las dudas existentes sobre aspectos concretos. El IC explicará el modelado propuesto, discutiéndose posibles mejoras y ampliaciones. La representación gráfica de las distintas versiones del algoritmo queda como testimonio de los avances y mejoras efectuados.

iv) Se repite el punto (ii) hasta encontrar una versión definitiva, pudiéndose entonces comenzar la fase de implantación. En muchos casos la fase de implantación impondrá o sugerirá cambios en el algoritmo, que serán discutidos conjuntamente y reflejados en la representación gráfica.

Como puede verse, este proceso sistemático comprende los tres bucles representados en la figura -3.1-: el bucle de adquisición, el bucle de modelado y el bucle de implantación. El sistema de representación formal permitirá también la transmisión del algoritmo y su especificación

informática, facilitando la implantación. Jugará el papel que juega el conjunto de planos que es imprescindible seguir en el diseño y construcción de cualquier sistema complicado de Ingeniería o Arquitectura.

En las sucesivas etapas de adquisición del conocimiento y de modelado, la representación formal del problema (y su método de resolución) irá transformándose paulatinamente. Comenzará con descripciones de grandes bloques para las funciones y nombres largos y descriptivos para las variables, y acabará asignando nombres concretos a las variables y entrando en detalles de arquitectura de programación (como puede ser la estructuración de datos y la gestión de memoria).

El algoritmo que se implantará en la máquina es la representación formal de la función que resuelve el problema, o más exactamente,de la ley de formación de valores de dicha función. Recordando que un algoritmo es un conjunto estructurado de procesos y símbolos, se llama aquí modelado del problema al proceso de creación del algoritmo encargado de resolverlo.

Modelar un problema es por tanto descomponerlo en un conjunto estructurado de subproblemas; cada subproblema es formalmente una función vectorial de variable vectorial, y se relaciona con los otros subproblemas por intercambios de datos (variables) y por las secuencias en que deben ser resueltos. Cada subproblema se modelará a su vez,

dando lugar a otro conjunto de subproblemas más elementales. El proceso de descomposición terminará cuando todos los subproblemas no modelados sean suficientemente elementales o ya conocidos.

El método general sugerido para modelar un problema, o lo que es igual, para desarrollar un algoritmo, consiste en el siguiente proceso de análisis y síntesis:

i) Se analiza la función global del problema, descomponiéndola en funciones más sencillas; cada una de estas funciones se analiza a su vez, hasta llegar a obtener funciones elementales o ya conocidas.

ii) Se estudia qué variables tiene cada función elemental como argumentos y qué variables son los valores (resultados obtenidos) de cada función.

iii) Se procede a la síntesis (unión, composición) de datos y funciones, buscando la coherencia del conjunto y agrupando los datos en árboles para que cada función, a cualquier nivel del árbol de funciones, tenga un número razonable de argumentos y valores.

iv) Los pasos (ii) y (iii) pueden revelar por un lado la necesidad de funciones auxiliares para transformar o inicializar variables, y por otra la existencia de funciones comunes y vectores de datos

comunes en distintas partes del algoritmo. Estas circunstancias dan paso a un segundo proceso de análisis-síntesis, y así sucesivamente hasta conseguir un modelo completo y satisfactorio del algoritmo (ver "bucle de modelado" en la figura -3.1-).

La tarea del Ingeniero del Conocimiento es el diseño e implantación de algoritmos capaces de resolver problemas, siguiendo cualquier método sistemático. Aquí se propone utilizar un sistema gráfico de representación del conocimiento para facilitar el proceso descrito, desde la adquisición del conocimiento hasta la implantación informática. El sistema de representación del conocimiento utilizado en todo el proceso puede ser a la vez un sistema de representación de programas con lo que el desarrollo de un algoritmo producirá a la vez la especificación informática correspondiente.

3.2.2- Algoritmo: variables, funciones y estructura de red de árboles

En la figura -3.1- se aprecia que los productos de las fases de adquisición del conocimiento y de modelado son un algoritmo y su representación gráfica. En este apartado se describe con cierto detalle el concepto de algoritmo y su estructura general.

Se pretende que la máquina sea capaz de resolver un problema. Ahora bien, la resolución de un problema es formalmente una función vectorial de variable vectorial definida entre los datos y las soluciones, cuya ley de obtención de imágenes se expresa por medio de un algoritmo.

En el proceso de resolución de un problema a cada valor particular del vector formado por los argumentos (datos del problema, variables de entrada, conjunto origen) le corresponde un valor del vector valor de la función (conjunto imagen), formado por una o varias soluciones del problema. El proceso de resolución de un problema es por tanto una función definida entre las variables de entrada y las de salida.

En cuanto a las variables, la definición de función entre conjuntos es más general que entre variables numéricas, pero en la práctica todo conjunto puede siempre representarse por una variable vectorial tal que a cada elemento del conjunto le corresponda un valor único de la variable y viceversa.

Se observará que el concepto de variable vectorial y el de función vectorial son suficientemente generales como para representar cualquier procedimiento de resolución de problemas. Los vectores tendrán los campos o elementos necesarios para almacenar toda la información que, sobre los entes que intervengan en el problema, sea utilizada en la resolución del mismo. Estos entes podrán ser de carácter abstracto, como

por ejemplo un "escenario económico" o unos "factores de peso de restricciones", o bien objetos reales como un motor o una central eléctrica.

No hay ninguna limitación teórica al nivel de abstracción en los razonamientos, pues las propias funciones pueden ser variables de entrada y de salida de otras funciones de "orden" superior: no hay que olvidar que en el ordenador las funciones o procesos se identifican por una variable, al igual que los datos. Al final del apartado 3.4. se trata ampliamente la relación entre la abstracción y la posible "creatividad" de una herramienta informática.

Las dos estructuras resultantes, la de variables y la de funciones, adoptan una forma general análoga: la de una red de árboles, esto es, la de un conjunto de árboles que comparten ciertas ramas. La figura -3.2- muestra un caso de tres árboles organizados según esta estructura.

La red de árboles de funciones tiene un sólo nodo raíz, que es la función que resuelve el problema. Esta es la función principal o, una vez implantada, el programa principal. Sin embargo, la red de árboles de variables tiene al menos dos nodos raíz, correspondientes a las variables de entrada y a las variables de salida. En la práctica se suelen considerar más de dos árboles de variables, pues la naturaleza del problema suele sugerir una clasificación inicial razonable de los datos.

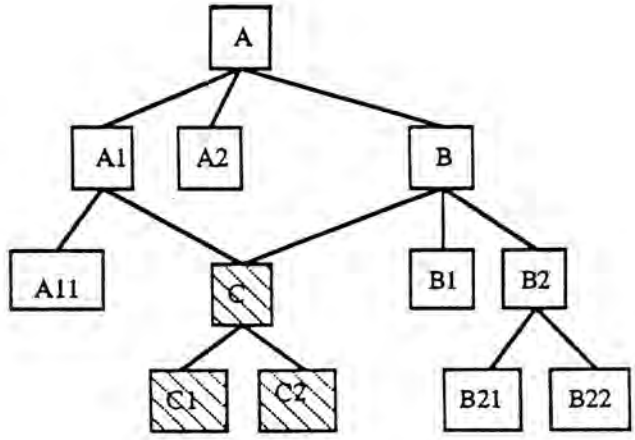


figura -3.2-

Estructura de red de árboles.

Un árbol de funciones está formado por un conjunto de ellas que sólo son llamadas por una función (son subproblemas de un sólo problema, tienen un sólo nodo padre), a excepción de la función principal (nodo raíz) que es llamada por más de una función, formándose así la red de árboles. En lenguaje informático clásico la raíz de un árbol de funciones se suele llamar subrutina.

Un árbol de variables está formado por un conjunto de vectores que son componentes de un sólo vector (estructuras de datos que son parte de una sola estructura de datos de nivel superior), a excepción del vector

(estructura) raíz del árbol, que aparece como componente de dos o más vectores de nivel superior, formándose así la red de árboles.

En la red de árboles de funciones, compartir la misma función supone un ahorro de código de lenguaje de programación y una simplificación en la estructura que aumenta la robustez del algoritmo. En el árbol de vectores de datos la ventaja de compartir ramas es evidente, dado el ahorro de memoria que representa el no duplicar la información sobre un mismo ente (no hay que confundir compartir el mismo tipo de almacenamiento y compartir el contenido: se puede ser del mismo tipo sin compartir el contenido pero no se puede compartir el contenido sin ser del mismo tipo). Representa además explícitamente el hecho de que la información manejada al resolver un problema complejo no se puede clasificar o dividir de una forma excluyente, ,porque los conjuntos de datos suelen presentar intersecciones importantes.

El modelado de un problema puede dar lugar a estructuras muy complejas de funciones y de variables. La utilidad de un sistema gráfico de representación formal salta a la vista, especialmente si se tiene en cuenta que el modelado es una producción sucesiva de versiones cada vez más refinadas de un algoritmo, lo que exige unos medios ágiles y fiables para mantener la coherencia de cada nueva versión.

3.2.3- Implantación informática

Una vez creado un algoritmo capaz de resolver un determinado problema, se podrá proceder a su implantación informática. La implantación informática de un algoritmo dado depende naturalmente de las características de dicho algoritmo; a su vez, la implantación informática debe tenerse en cuenta a la hora de concebir el algoritmo si se desea obtener buenos resultados. En cualquier caso, la implantación se enfrenta a tres problemas fundamentales: las limitaciones de tiempo, las limitaciones de memoria y la aparición difícilmente evitable de errores de programación.

La aparición de errores de programación se puede reducir con ayuda de una buena representación formal del algoritmo, a la vez que se facilita su posterior depurado. Los problemas de tiempo y memoria surgen de las limitaciones de las máquinas, y normalmente pueden corregirse introduciendo mejoras en el algoritmo: éste es el significado del " bucle de resultados " que aparece en la figura -3.1-.

La implantación se podrá llevar a cabo sistemáticamente comenzando por las funciones elementales; una vez que todas y cada una de las funciones que forman una función más compleja se hayan implantado y comprobado, se integrarán para formar dicha función compleja, generando así progresivamente toda la estructura de funciones.

Cada vez que se integre un grupo de funciones habrá que comprobar los resultados de la función conjunta, si es necesario con herramientas que simulen el comportamiento del exterior frente a dicha función; el tiempo invertido en la creación de estos "bancos de prueba " suele ahorrar largas sesiones de depurado al final del proyecto.

Para que la implantación sea eficaz es preferible que el modelado del problema, y por tanto el algoritmo, se haya orientado hacia una arquitectura informática y un lenguaje determinados. En el campo de la Ingeniería del Conocimiento se encuentran aplicaciones informáticas implantadas en una gran variedad de lenguajes de programación. Es frecuente también que se desarrollen lenguajes a medida para aplicaciones especiales, aunque habitualmente sean versiones especializadas de lenguajes ya existentes.

La existencia de tal variedad de lenguajes aplicados a un mismo campo, el de la Ingeniería del Conocimiento aplicada al diseño, muestra la dificultad que entraña pronunciarse con objetividad a favor de un lenguaje de alto nivel. La selección de un " sistema concha " determinado es aún más difícil, puesto que la formalización de cada problema y su resolución son muy subjetivas.

El ser humano utiliza para comunicarse lenguajes textuales (naturales) y lenguajes gráficos. Estos últimos son especialmente importantes en actividades como la Ingeniería, y por tanto también

debieran serlo en la Ingeniería del Conocimiento. Por otro lado la máquina opera sólo con variables y funciones (algoritmos) y requiere una gestión eficaz de los recursos de tiempo y memoria, lo que sugiere usar lenguajes de bajo nivel para diseñar el algoritmo a medida de la máquina.

Lo que aquí se propone es una combinación de ambos extremos: diseñar y especificar los algoritmos en un lenguaje gráfico-textual e implantar luego el resultado en un lenguaje de bajo nivel. Dada una especificación completa y consistente, la fase de implantación puede ser una tarea rutinaria, limitándose la creatividad al aprovechamiento óptimo de los recursos de la máquina.

Esta combinación de lenguajes gráficos y lenguajes de bajo nivel es especialmente atractiva de cara al futuro, si se piensa que actualmente ya se desarrollan herramienta informáticas que se comunican con el usuario por lenguajes gráficos y son luego capaces de generar código automáticamente en distintos lenguajes de bajo nivel, optimizando incluso los recursos de la máquina automáticamente. La metodología aquí propuesta es la compuesta por la metodología ANA (apartado -3.3-) como lenguaje gráfico de alto nivel y el C como lenguaje de bajo nivel (apartado -3.4-).

3.3- Representación del conocimiento: metodología ANA

Ya se ha establecido que es necesario un sistema gráfico de representación formal para la adquisición del conocimiento, el modelado y la implantación de un sistema automático de resolución de problemas. También se ha establecido que el resultado del modelado de un problema es la creación de dos estructuras en forma de red de árboles, la de funciones y la de variables.

El sistema de representación elegido debe ser sencillo y completo, capaz de representar fielmente y sin ambigüedad un algoritmo a cualquier nivel de detalle que se desee y en cualquier nivel de modelado o descomposición en que se encuentre (recuérdese el proceso de análisis-síntesis ya descrito). En otras palabras, y siguiendo la línea de ideas de este trabajo, debe ser capaz de representar cualquier versión tanto de la red de árboles de funciones como de la red de árboles de variables de un algoritmo. El sistema de representación elegido aquí es el método ANA.

Es para poder utilizar esa metodología de manera eficiente que se ha desarrollado la herramienta informática gráfica objeto de este proyecto. El método de representación esta descrito en el capítulo 5.

Para comprobar la capacidad de representación de ANA habrá que estudiar qué puede aparecer en un algoritmo. Nos limitaremos al estudio

de las relaciones que pueden aparecer en la estructura de funciones y la estructura de variables de un algoritmo. Se distinguen primero dos grupos de relaciones: las estáticas y las dinámicas, que se corresponden con lo que en literatura especializada se suele denominar conocimiento estático y conocimiento dinámico de un problema.

Las relaciones estáticas en un algoritmo genérico son las mismas tanto para la estructura de variables como para la estructura de funciones, ya que son simplemente las propias de la teoría de conjuntos (cada conjunto es un nodo de la red de árboles) o del álgebra de clases: Identidad, Pertenencia, Complementariedad, Unión e Intersección. El método ANA es capaz de representar sin dificultad todas ellas para los dos tipos de estructuras.

Las relaciones dinámicas son más complejas de enumerar y definir. Son distintas para la estructura de variables y la estructura de funciones, y tal vez las listas de relaciones dinámicas que se presentan a continuación no sean exhaustivas.

En lo referente a la estructura de funciones se pueden enumerar las siguientes relaciones dinámicas, que definen los mecanismos de control según los cuales se aplica en cada momento una de las diversas subfunciones que forman una función:

- **Secuencia**, u orden relativo en que se aplican las diversas funciones.

- **Iteración**, o repetición de una función o grupo de funciones hasta cumplir un cierto objetivo.

- **Recursión**, cuando al modelar un problema determinado éste vuelve a aparecer en su misma forma, y así sucesivamente hasta cumplir un cierto objetivo. En la mayoría de los casos la recursión puede sustituirse por una iteración, pero conceptualmente son procesos distintos que a veces no resultan intercambiables.

- **Decisión** de " cuál función se aplica ", condicionada a la certeza o falsedad de un grupo de asertos. Cuando al modelar un problema aparece este tipo de relación, la función que resuelve el problema se dirá que es una función redefinida (ver Rodriguez-Piñeiro).

- **Inicialización**, cuando una función establece las condiciones de partida para que otra función o grupo de funciones pueda aplicarse. Usualmente esto es necesario con procesos iterativos y recursivos, que requieren tanto unas condiciones iniciales como otras condiciones para reconocer el final del proceso.

Con respecto a la estructura de variables se pueden distinguir las siguientes relaciones dinámicas, dependientes del flujo de información de un algoritmo y por tanto de la estructura de funciones (por lo que la estructura de variables debe definirse a partir de la de funciones, y no al revés):

- **Causa-efecto** (o también implicación, argumento-valor, origen-imagen, entrada-salida) entre individuos de la misma o distintas clases, o sea una relación del tipo " ser función de ".

- **Instanciación o ejemplificación** de una clase, esto es, tomar un punto concreto de entre todos los puntos posibles de un espacio vectorial. Como casos particulares de ejemplificación pueden citarse la inicialización o selección inicial, y la sustitución de un individuo por otro de la misma clase, es decir, de un punto de un espacio vectorial por otro punto del mismo espacio.

- **Descomposición** de una clase en subclases, es decir, descomposición de un espacio vectorial en subespacios. La relación inversa será de **composición** o suma.

La metodología ANA es capaz de representar sin ambigüedades todas estas relaciones: lo que se puede representar, se puede programar, y viceversa. Utilizando un lenguaje propio de la programación, puede representar el desglose de una estructura (o vector) de variables en sus componentes o elementos, al igual que la operación inversa; la concesión de valores concretos a los argumentos de la funciones o subrutinas; el hecho de igualar un vector o estructura de variables a otro del mismo tipo; y, por supuesto, la relación funcional clásica de "argumentos de entrada-valores de salida " de las funciones.

En cuanto a las secuencias de actuación de las funciones que forman el algoritmo, este método es capaz de representar conjuntos complejos de bucles, bloques "IF", recursiones e inicializaciones, mezclados naturalmente con conjuntos de acciones consecutivas. La gran diferencia con los diagramas de flujo tradicionales consiste en que todas las relaciones dinámicas, tanto las de variables como las de funciones, aparecen representadas en el mismo diagrama. Ya se verá que esto, junto a la absoluta modularidad de la representación ANA, representa un salto cualitativo importante para la comprensión, depurado y desarrollo sistemático de los algoritmos.

La representación ANA de un algoritmo es en principio independiente de su implantación, pues refleja sólo la estructura de subproblemas y la estructura de variables que aparece al analizar sistemáticamente dicha estructura de subproblemas, es decir, representa el modelado de la resolución de un problema en forma de algoritmo. Sin embargo, un modelado que pretenda producir un algoritmo eficaz deberá desarrollarse a medida del estilo y del lenguaje de programación que se vayan a usar en la implantación.

Aunque no se menciona la posibilidad de que varias funciones se apliquen simultáneamente, es decir, no se describe el procesamiento en paralelo, ANA es capaz de representar esto perfectamente. Sin embargo, desde un punto de vista teórico no hay diferencia entre realizar varias acciones en paralelo o secuencialmente, aunque en la práctica signifique

un ahorro de tiempo. El procesamiento en paralelo se considera aquí un detalle arquitectural y no determinante de un algoritmo, aunque la posibilidad o imposibilidad de realizarlo puede sugerir a veces diseños radicalmente distintos de un mismo algoritmo.

Entre los dos grandes estilos de programación actuales, la Programación Estructurada y la Programación Orientada al Objeto el primero parece encajar más fácilmente con el método de desarrollo y el sistema de representación que aquí se propone. En efecto, la Programación Estructurada está basada en procedimientos modulares organizados según una estructura de red de árboles como la que aquí se describe, por lo que la representación ANA puede usarse directamente como especificación informática de un algoritmo.

La Programación Orientada al Objeto se basa en el concepto abstracto de "objeto", siendo cada "tipo de objetos" lo que en Lógica se llama clase. En el código de los programas escritos en lenguajes de este estilo todas las variables y procedimientos relacionados con un determinado tipo de objetos aparecen agrupados en la declaración de dicho tipo. Esto es bueno en cuanto a mantener la coherencia de cada objeto en su propio entorno, pero en cambio no es fácil reconocer la secuencia de acciones de un programa ni prever las consecuencias de un cambio en esa secuencia.

En Programación Orientada al Objeto, el método ANA sería de gran ayuda para diseñar los algoritmos y ayudar al desarrollo del código, pero en su versión actual no serviría directamente como especificación informática, pues no representa lazos de unión intrínsecos entre variables y funciones.

3.4- Lenguaje de programación C

Aunque la selección del lenguaje es una decisión personal, se van a exponer aquí razones más o menos objetivas para proponer el lenguaje C como el más adecuado para desarrollar proyectos complejos de Ingeniería del Conocimiento. Se excluyen de la discusión los lenguajes construidos a medida de una aplicación concreta, pues se da por supuesto que siempre serán preferibles en su propio contexto a cualquier lenguaje de aplicación general.

Veamos una serie de razones para preferir el C sobre otros lenguajes, ya sea para "Inteligencia Artificial" como para programación convencional:

- El C proporciona todos los recursos que necesita el programador más exigente para implantar algoritmos complejos: estructuras de datos definibles por el usuario, memoria dinámica, recursión y manejo de direcciones de memoria como otro tipo cualquiera de variables (punteros).

- En los lenguajes más " rígidos ", como el FORTRAN, todas las ligaduras (bindings) de las variables con sus campos de memoria se realizan en tiempo de preproceso. En los lenguajes más " flexibles ", como el LISP, todas las ligaduras se realizan en tiempo de proceso. En C es el propio programador el que elige con absoluta libertad cuándo se producen y cuándo se liberan estas ligaduras, optimizando así el producto en cuanto a tiempo y memoria.

- El C es actualmente el lenguaje más compatible y popular, y con mejores perspectivas para el futuro, a juzgar por su capacidad de difusión; y son evidentes las ventajas que siempre acompañan al hecho de unificar lenguajes.

- En la práctica, muchos compiladores de lenguajes tales como LISP y Prolog están escritos en C, con la ineficacia propia de tener que traducir un estilo declarativo a otro procedural (por ejemplo, de Prolog a C).

- Los paquetes de programas auxiliares, tales como editores, gráficos e interfaces están escritos en C, lo que acarrea frecuentes problemas de encadenado si el programa principal no está en C.

- El C proporciona un esqueleto estructurado a los programas sin limitar la creatividad, facilitando la consistencia de aplicaciones que además resultan ser de una rapidez espectacular.

Las razones anteriores se pueden resumir en la siguiente afirmación: cuando se necesita controlar todos los recursos de la máquina con funciones rápidas, seguras y eficaces se recurre a un lenguaje de bajo nivel como el C. El inconveniente principal del C es precisamente su bajo nivel , es decir, su proximidad al lenguaje máquina y consiguiente falta de parecido con el lenguaje humano, que hacen al código escrito en C difícilmente comprensible.

En defensa de los lenguajes de bajo nivel se puede argumentar que los de alto nivel sólo facilitan una comprensión muy limitada de los programas, pues sigue siendo muy difícil obtener una visión de conjunto contando sólo con el código fuente, por muy de alto nivel que sea el lenguaje. Sin embargo, con un sistema potente de representación como el ANA, se puede facilitar significativamente la comprensión y especificación informática de un programa.

Hay además un argumento realista y más concluyente de cara al futuro por el cual un lenguaje de bajo nivel es preferible: es evidentemente más fácil y eficaz que un ser humano descienda al nivel de lenguaje de la máquina en vez de que ésta ascienda al nivel de lenguaje de un ser humano. La existencia de bibliotecas ("libraries") de programas cada vez más sofisticados y agrupados por especialidades de aplicación es la que debe facilitar el trabajo de los programadores, sin limitar en absolutó sus recursos y creatividad personal.

En el campo de la Ingeniería del Conocimiento hay razones adicionales para preferir utilizar lenguajes de bajo nivel. Una de ellas es la velocidad de ejecución, que en la resolución de problemas complejos puede significar la rentabilidad o no de la inversión en un proyecto.Otra es la ya mencionada integración con herramientas sofisticadas de gráficos y demás interfaces "amigables". Pero la más importante de cara a grandes proyectos es que el grado de flexibilidad estructural lo puede escoger el programador.

Es siempre deseable que el programador intente crear una estructura general rígida y aislar los módulos que realmente necesitan ser flexibles; esto es importante porque la flexibilidad aumenta la dificultad para detectar y corregir errores, y a la vez hace más lenta la ejecución. No debe olvidarse el papel tan destacado que juega la fase de depurado en un proyecto informático.

Como resumen de todas estas consideraciones, se pude enunciar lo siguiente: lo que se necesita en la Ingeniería del Conocimiento no es un lenguaje sofisticado, sino unas técnicas de modelado sofisticadas que puedan generar representaciones suficientemente abstractas de los problemas, auxiliadas por un sistema gráfico de representación claro y potente y a ser posible común para algoritmos abstractos e implantaciones informáticas. El contar con un sistema gráfico de representación formal ha permitido una tarea formidable de creación y

desarrollo en disciplinas tan complejas como la Ingeniería y la Arquitectura.

Capítulo 4

APLICACIÓN DE LA METODOLOGÍA 'ANA' A ESTE PROYECTO

Aunque en un principio parezca que existen tres fases bien distintas (Adquisición del conocimiento, modelado e implantación) en la realización de un proyecto, estas se entremezclan mucho debido a los bucles de realimentación vistos en la figura -3.1-. En este capítulo vamos a intentar separar en lo posible esas tres fases para cada versión.

4.1- Versión 1

A continuación se detallan las distintas fases de la versión 1.

4.1.1- Adquisición del conocimiento

Lo primero que hay que ver cuando se empieza un proyecto es "qué hay que hacer". Para ello hubo una fase de aprendizaje del lenguaje gráfico ANA, sobre en todo en cuanto a la representación dinámica siendo esta la parte más complicada y sobre todo siendo esta la única que se planteaba introducir en la primera versión de la herramienta.

La documentación que tenía para ello era básicamente un programa que representaba en pantalla un diagrama ANA básico y la ayuda del experto en la materia (el supervisor de este proyecto que fue a su vez el que desarrolló la metodología ANA).

El estudio del programa forma también parte de otra fase de la adquisición del conocimiento: "cómo hay que hacerlo ". Para ello se estudió una versión prototipo desarrollada previamente. Ello permitió además adquirir conocimientos más elaborados del lenguaje C que no se dominaba. Solo se tenía conocimientos teóricos de un cursillo realizado.

Se llegó así al estudio de que es lo se quiere que el usuario vea y pueda hacer. Lo que se quería en esta herramienta era tener acceso a todas sus opciones por medio de menús. Hubo que estudiar como conseguir menús profesionales en una pantalla gráfica de PC.

4.1.2- Modelado

Para hacer el modelado de este proyecto no se hicieron casi planos del algoritmo ya que se carecía de una herramienta adecuada: la misión del proyecto consistía en elaborar esa misma herramienta.

Por ello se orientó muy rápidamente a la implantación directa del algoritmo, con todos los problemas que ello acarrea. Es importante mencionar el hecho de que hubo que remodelar las estructuras de datos

sobre las que se basaban los diagramas ANA. Pero esto fue sólo un remedio provisional para obtener resultados a corto plazo. Por ello surgió posteriormente la idea de hacer una segunda versión nueva partiendo desde cero.

4.1.3- Implantación

La implantación de la herramienta se hizo en lenguaje C con el compilador de la casa Borland llamado "Turbo C". Las razones de elegir este lenguaje están ya expuestas en el apartado 3.4.

A pesar de no haber modelado el programa con el lenguaje gráfico ANA se intento seguir su método de implantación de manera de hacer árboles de funciones y de variables. Las funciones se llaman por el nombre de la función raíz del árbol más una serie de números que indican la profundidad en la que se encuentran y a que rama pertenece. De esta manera se sabe en todo momento a que función padre pertenece cada subfunción que a su vez puede ser padre de otras. En cuanto a las variables se organizaron en grandes estructuras que juntaban todos los datos que estuviesen relacionados entre sí. Por ejemplo un diagrama ANA estaba totalmente descrito con una sola estructura de datos.

Constantemente, a lo largo de la implantación de la herramienta hubo que realimentar el proceso y volver a adquirir conocimientos sobre que se quería hacer y como implantarlo. Se presentaron sobre todo

muchos problemas de que se permitiría hacer en la herramienta: ello suponía una implantación muy distinta en unos casos y en otros.

En el Anejo I podemos ver una selección de código de esta versión. No se ha incluido todo debido a su magnitud (50.000 líneas).

4.2- Versión 2

A continuación detallamos los distintos pasos de la segunda versión.

4.2.1- Adquisición del conocimiento

Gran parte de esta fase ya estaba hecha. Se puede decir que la realización de la primera versión fue una gran ayuda para saber que se quería en la segunda. En realidad se planteó hacer una segunda versión debido justamente a todo el conocimiento adquirido anteriormente: ya sabíamos a que problemas había que enfrentarse.

Pero no se deseaba limitarse a hacer una versión mejorada sino que se quería aportar algo nuevo: un entorno donde se podría representar las relaciones estáticas de la misma manera que las dinámicas.

Además, habiendo aprendido ya de la versión anterior el problema de la realimentación en la implantación, se quería tener ya hechos todos

los árboles de menús para los cuales se hizo un estudio de qué se quería que hiciese la herramienta en sus más mínimos detalles.

4.2.2- Modelado

En esta versión el modelado es mucho más completo. Aquí ya se dispone de una herramienta que nos permite diseñar y especificar el programa desde un principio. En el apartado de "Planos" (Documento nº2) se puede ver una selección de los diagramas ANA realizados hasta ahora para esta segunda versión.

La primera labor, sin embargo, fue el diseño de las nuevas estructuras partiendo de los conocimientos adquiridos durante el desarrollo de la primera versión. Estas son mucho más consecuentes con lo que se les pide. El diseño de estas nuevas estructuras sobre las que se basa todo el programa fue la razón principal de una segunda versión.

Se ha diseñado la herramienta totalmente modular de manera que ya se tienen algunos módulos totalmente diseñados y especificados. Dada la magnitud del proyecto, no se ha podido finalizar esta fase para toda la herramienta.

4.2.3- Implantación

La implantación que se ha hecho de esta versión es muy reducida. Se han seguido las mismas pautas que en la primera versión salvo que esta vez se ha querido hacer en Standard C, utilizando el mismo compilador que en la primera versión.

En lo que se lleva hecho los cambios aportados al modelo hecho anteriormente se han reducido considerablemente respecto a la otra versión. Solo ha habido que modificar detalles sin importancia que no alteran en nada la estructura general de lo diseñado hasta ahora

Se puede ver en el anejo II una selección del código implantado hasta ahora. No se incluye todo porque se desea que en el futuro esta herramienta pueda comercializarse.

Capítulo 5

LENGUAJE GRÁFICO ANA

Este proyecto consiste en la elaboración de una herramienta que nos permita diseñar y especificar un programa mediante un lenguaje gráfico. En este capítulo presentamos las bases del lenguaje ANA, que es una adaptación de hardware hacia software de la metodología CORE (COntrolled Requirements Expression: ver en bibliografía Systems Designers).

5.1- Introducción

La Ingeniería del Conocimiento se considera aquí como la aplicación de los conocimientos de Lógica al desarrollo de sistemas automáticos de resolución de problemas. La resolución de un problema es formalmente una función entre dos espacios vectoriales, el de datos (variables de entrada) y el de soluciones (variables de salida), según muestra la figura -5.1-.

La resolución de un problema se realiza mediante un algoritmo, que formalmente es la ley de obtención de imágenes de la función "resolución del problema".

Un algoritmo consta de dos estructuras estrechamente relacionadas, la de variables y la de funciones. Ambas estructuras responden a la forma general de una red de árboles, es decir, un conjunto de árboles que comparten ciertas ramas.

figura -5.1-

La resolución de un problema como una función vectorial de variable vectorial.

El desarrollo de un algoritmo consiste en una serie de procesos sistemáticos de análisis y síntesis aplicados a la función original y a los espacios vectoriales de datos y soluciones. El objetivo es especificar por completo la red de árboles de funciones y la red de árboles de variables, así como las relaciones existentes entre ellas.

5.2- Relaciones estáticas: estructura de red de árboles.

Las relaciones estáticas son las que construyen la estructura de red de árboles, tanto en el caso de las variables como en el de las funciones. Una estructura compleja de variables consta de estructuras de variables más elementales; dicho de otra forma, un espacio vectorial consta de subespacios vectoriales. En el caso de las funciones, una función compleja (resolución de un problema, proceso) se compone en general de funciones más elementales (subfunciones, subprocesos, resolución de subproblemas).

Un árbol es una estructura rígida e indivisible que puede aparecer más de una vez en la red de árboles. El nodo raíz representa una estructura compleja de variables (o funciones); los nodos terminales del árbol pueden ser variables (o funciones) elementales o bien nodos raíz de otros árboles, constituyéndose así la red de árboles.

La figura -5.2- muestra la representación individual de tres árboles (A, B, C), al igual que su representación conjunta como red de árboles. Esta última no siempre será posible en su totalidad, pues la red de árboles puede ser demasiado compleja.

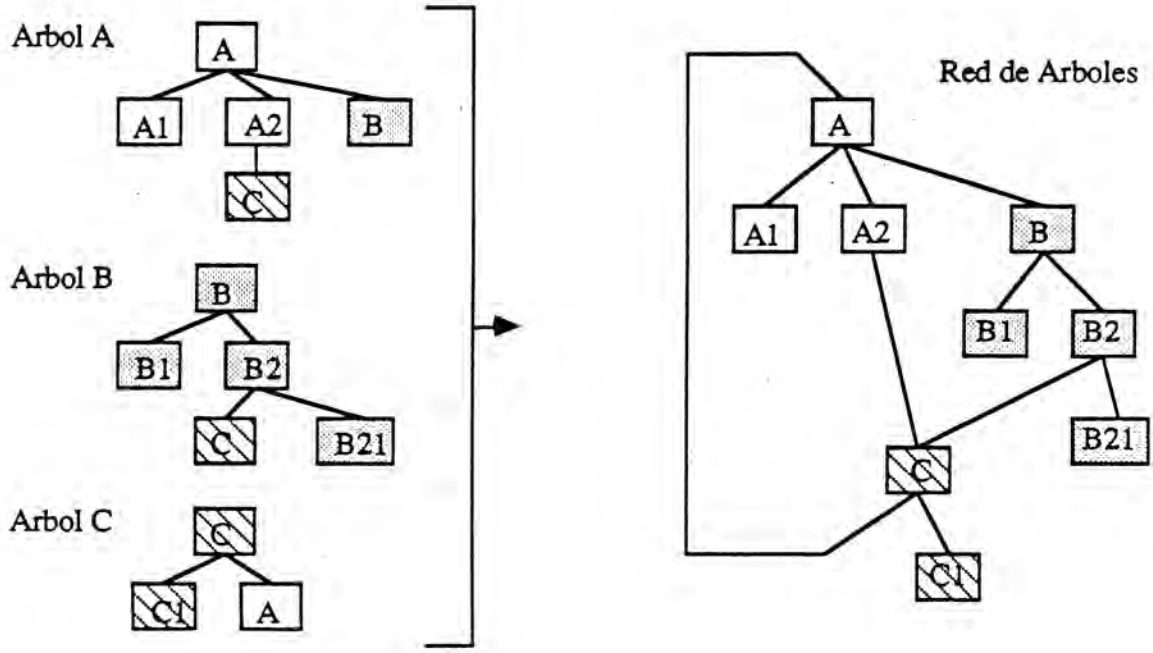


figura -5.2-

Representación de relaciones estáticas: red de árboles.

Es interesante observar que la red de árboles de la figura -5.2- es recursiva, pues el árbol A aparece como parte de C, y a su vez C aparece como parte de A.

5.3- Relaciones dinámicas: diagramas ANA

La representación de las relaciones estáticas es importante para organizar sistemáticamente y mantener la coherencia del conjunto de elementos de un problema. Sin embargo las relaciones dinámicas son las que describen el funcionamiento real del algoritmo encargado de resolver dicho problema, y por tanto su representación es indispensable para el desarrollo controlado de un proyecto de Ingeniería del Conocimiento.

Cada diagrama ANA muestra la estructura interna de funciones de una función pública (caja compuesta en las figuras) y sus relaciones dinámicas, que consisten en intercambios de variables y secuencias de actuación. La función pública es aquella que se puede llamar desde cualquier lado: tiene una identidad propia. La figura -5.3- representa la estructura interna de la función pública A (ver figura -5.2-, donde se muestran las relaciones estáticas), en la que aparecen tres subfunciones: una función pública A2, una función privada A1 (función simple en las figuras), ambas formando parte del árbol A, y una función pública B que es la raíz de otro árbol distinto.

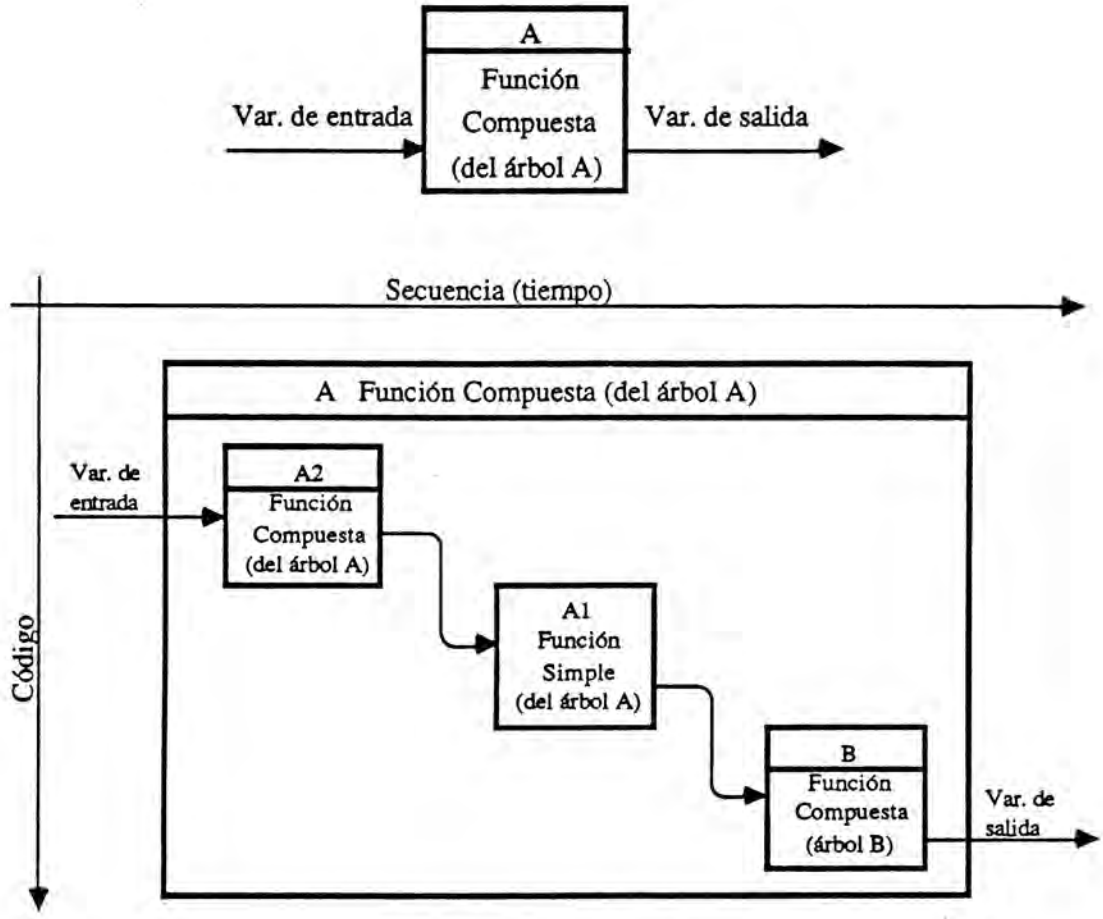


figura -5.3-

Diagramas ANA.

Una función se representa por una caja con dos textos en su interior: en la parte superior aparece el identificador (A1, A2, etc..) y en el inferior el nombre, que es una frase que explica la misión de la función. En las funciones públicas el identificador y el nombre aparecen

separados por una línea que parte la caja en dos; esto indica que es una función a la que se puede acceder desde cualquier parte del programa. Las funciones privadas (cajas simples en las figuras) son funciones que solo pertenecen a esa función descrita por el diagrama ANA en cuestión y no tienen separación entre el identificador y el nombre. Estas podrán ser descritas a su vez en ese mismo diagrama.

El orden horizontal de las cajas representa la secuencia en que actúan las funciones. El orden vertical indica la secuencia en que están escritas en el lenguaje de programación empleado. Lógicamente ambas secuencias suelen coincidir, por lo que las cajas aparecen normalmente en diagonal; la única excepción se presenta en el caso de acciones alternativas (o bloques IF) como se verá más adelante o en procesamiento en paralelo.

5.3.1- Relaciones dinámicas entre variables

Las relaciones dinámicas entre variables son tres: ejemplificación (o instanciación), que incluye la inicialización y la sustitución; la composición, y su inversa la descomposición; y por último la dependencia (ser función de) , que es la más importante. A continuación se explican estas relaciones y se muestra cómo se representan según el método ANA:

i) Ejemplificación: Una variable toma un valor concreto, lo que en Lógica equivale a seleccionar un individuo concreto de entre todos los individuos que componen una clase. Casos particulares de ejemplificación son la inicialización (figura -5.4.a-) y la sustitución (figura -5.4.b-).

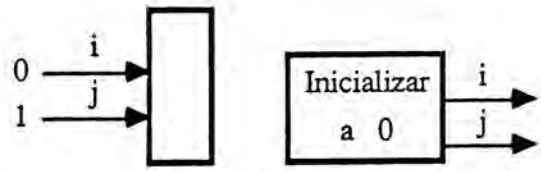


figura -5.4.a-

Dos formas alternativas de representar inicializaciones.

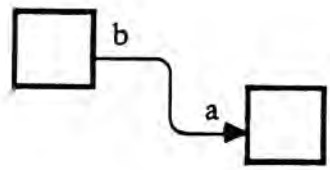


figura -5.4.b-

Sustitución de un valor por otro.

ii) Composición (y descomposición): Una estructura de variables se separa en sus partes y viceversa (figura -5.5-).

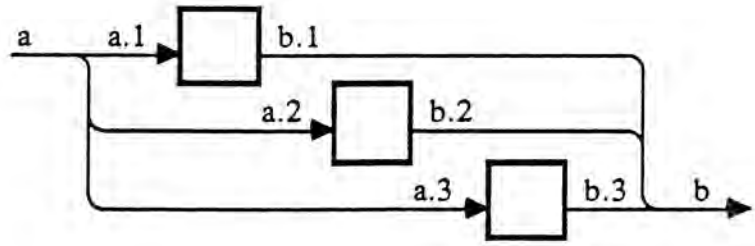


figura -5.5-

Composición de la variable b y descomposición de la variable a.

iii) Dependencia (ser función de): Es la relación más importante y básica. Indica que el valor de un conjunto de variables puede ser hallado a partir del valor de otras por medio de una determinada función. La figura -5.6- muestra cómo se representa esta relación en un caso en que dos variables (a, b) dependen de otras dos (c,d).

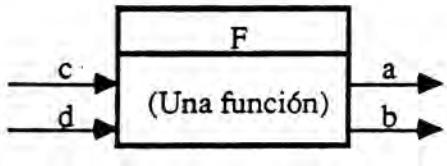


figura -5.6-

Dependencia entre variables.

Un aspecto muy importante en la fase de programación es conocer el "periodo de vida" (lifetime) de una variable a la que se desea asignar memoria dinámicamente. En la figura -5.7- se muestra la asignación dinámica de memoria para un vector A de n elementos, incluyendo la posterior liberación de dicha memoria.

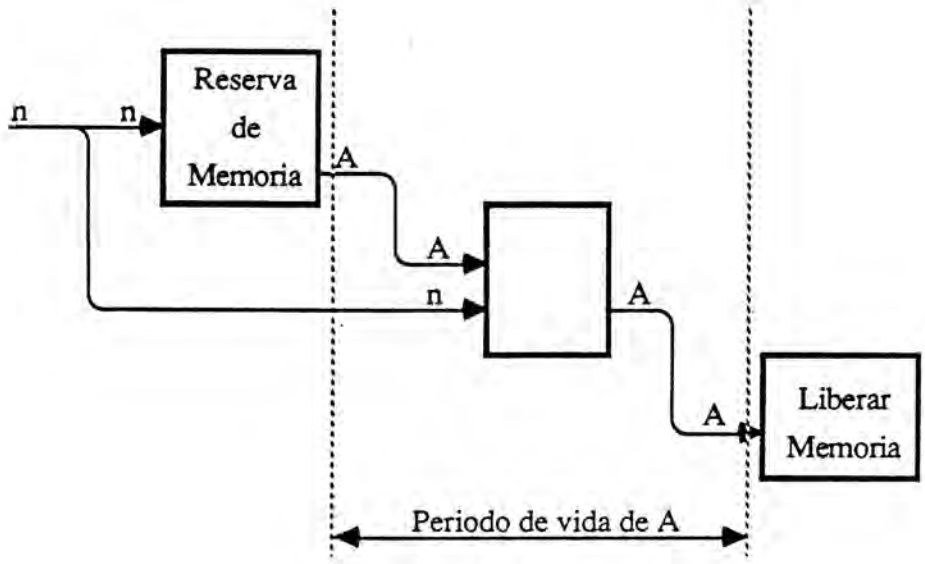


figura -5.7-

Asignación dinámica de memoria y periodo de vida de una variable.

Existen otro tipo de variables para las cuales no se reserva memoria sino que sencillamente se las declara. Estas son variables internas a la función. Una posible manera de hacer aparecer esa declaración es incluyendo en primera posición del diagrama una función privada que las describa. Tenemos un ejemplo de ello en la figura -5.8-.

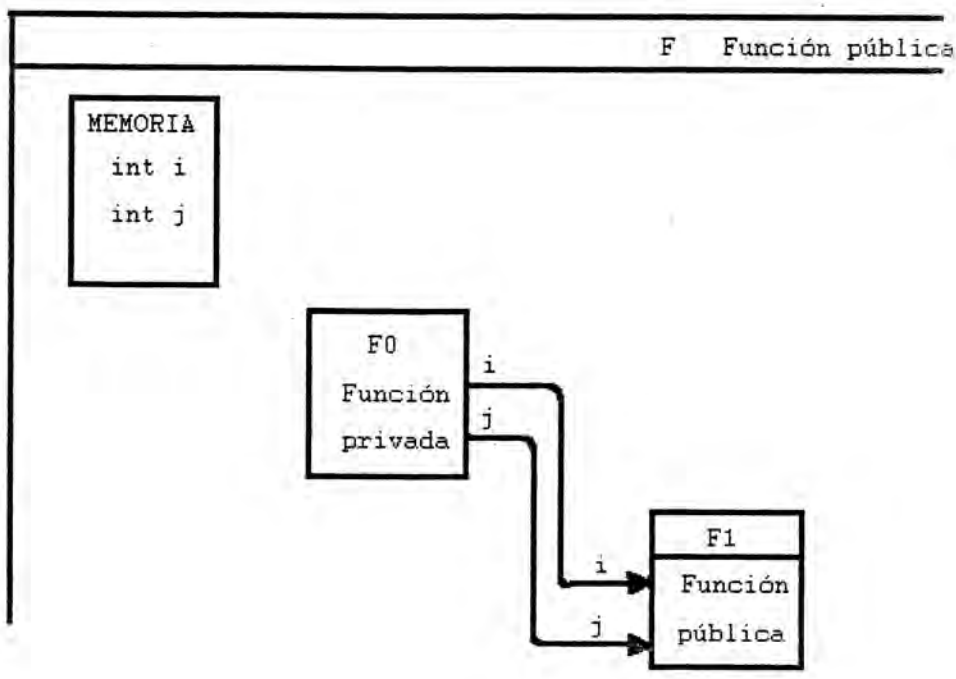


figura -5.8-

Declaración de variables internas.

5.3.2- Relaciones dinámicas entre funciones

Las relaciones dinámicas entre funciones son cuatro: la secuencia de actuación; la decisión entre alternativas (funciones condicionadas, bloques IF); la iteración, o bucles; y por último la recursión. A continuación se explican estas relaciones y su representación según el método ANA:

i) Secuencia: Es el orden relativo en que actúan las diferentes subfunciones que forman una función pública (figura -5.9-).

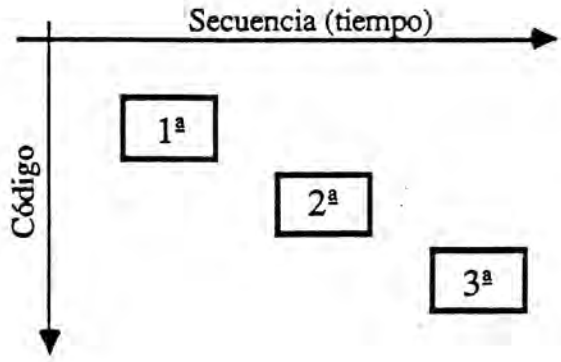


figura -5.9-

Secuencia de funciones.

ii) Decisión (o funciones alternativas, funciones condicionadas, bloque IF): según el valor en curso de ciertas variables se aplican unas funciones u otras.

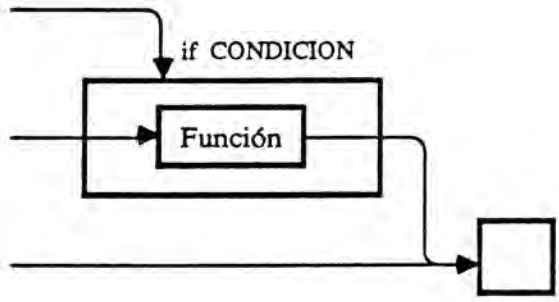


figura -5.10.a-

Función condicionada.

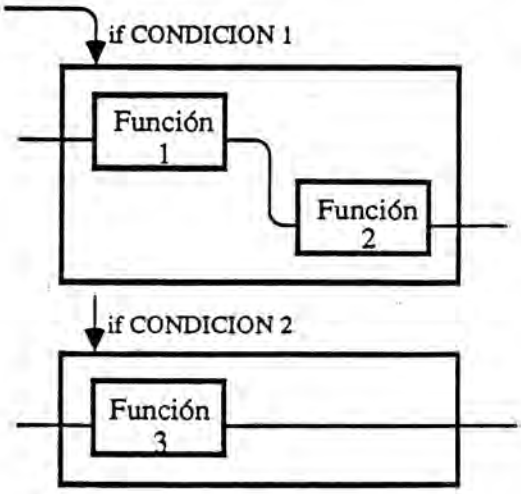


figura -5.10.b-

Bloques de funciones alternativas.

La figura -5.10.a- muestra una función que sólo se aplica en caso de cumplirse una cierta condición, mientras que la figura -5.10.b- representa dos bloques de funciones alternativas; estos bloques de funciones también se llaman macros en la primera versión de la herramienta y funciones privadas en la segunda.

iii) Iteraciones (o bucles): una iteración es una secuencia de funciones que se aplica repetidamente hasta que se cumple una cierta condición. De las variables que intervienen en el bucle algunas permanecen constantes, mientras que otras pueden modificar su valor en cada iteración. Estas últimas se llaman

orígenes y extremos del bucle, y " circulan" por la línea de retorno del bucle. Todo bucle tiene una condición de fin de bucle, que puede estar situada al principio o al final de cualquier función perteneciente al mismo. Justo debajo de la condición y escrito sobre la línea de retorno hay un texto auxiliar que ilustra el comportamiento o la finalidad del bucle.

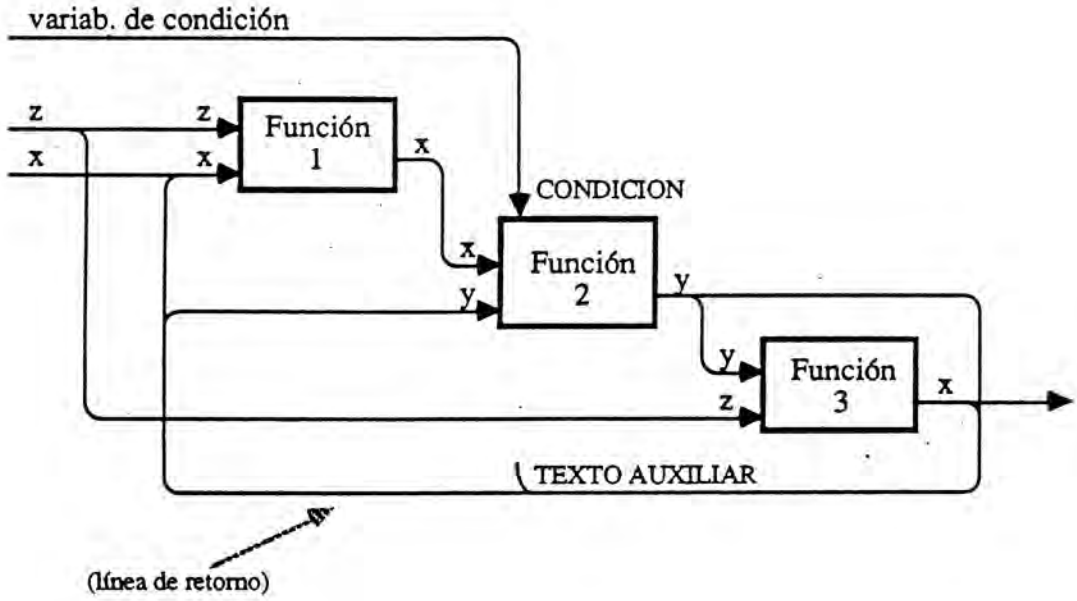


figura -5.11-

Ejemplo de representación de un bucle.

La figura -5.11- muestra un bucle que repite una secuencia de tres funciones, en el que la condición de fin de bucle se comprueba al principio de la segunda función. Las variables (x, y) se modifican en cada iteración, pero la variable z permanece constante durante todo el proceso.

En caso de que se desee más de una condición de fin de bucle se pueden añadir funciones condicionadas de salida de bucle, como la que aparece en la figura -5.12.a-. Si ningún dato es modificado en cada iteración la línea de retorno se representa como muestra la figura -5.12.b-.

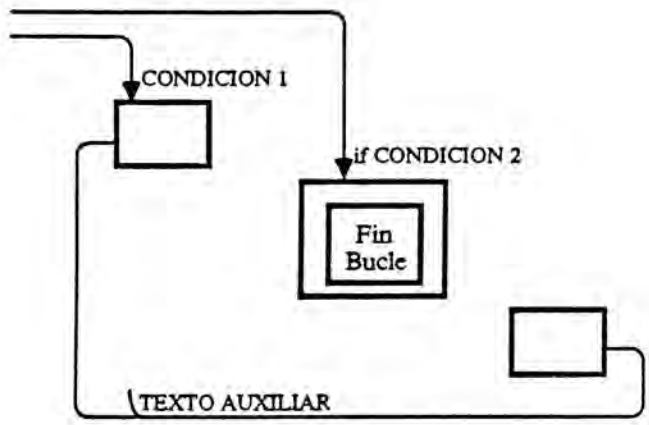


figura -5.12.a-

Condición adicional de salida de bucle.

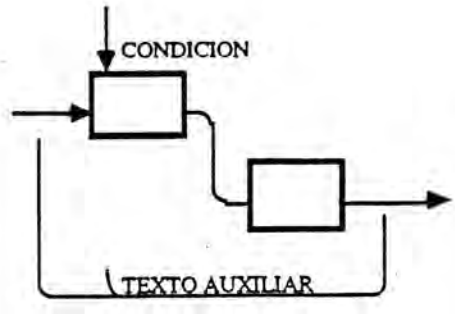


figura -5.12.b-

Línea de retorno cuando ningún dato se actualiza en cada iteración.

iv) Recursión: la recursión se produce cuando una función aparece como parte de sí misma en algún nivel de su análisis (ver apartado 5.2. Relaciones estáticas: red de árboles).

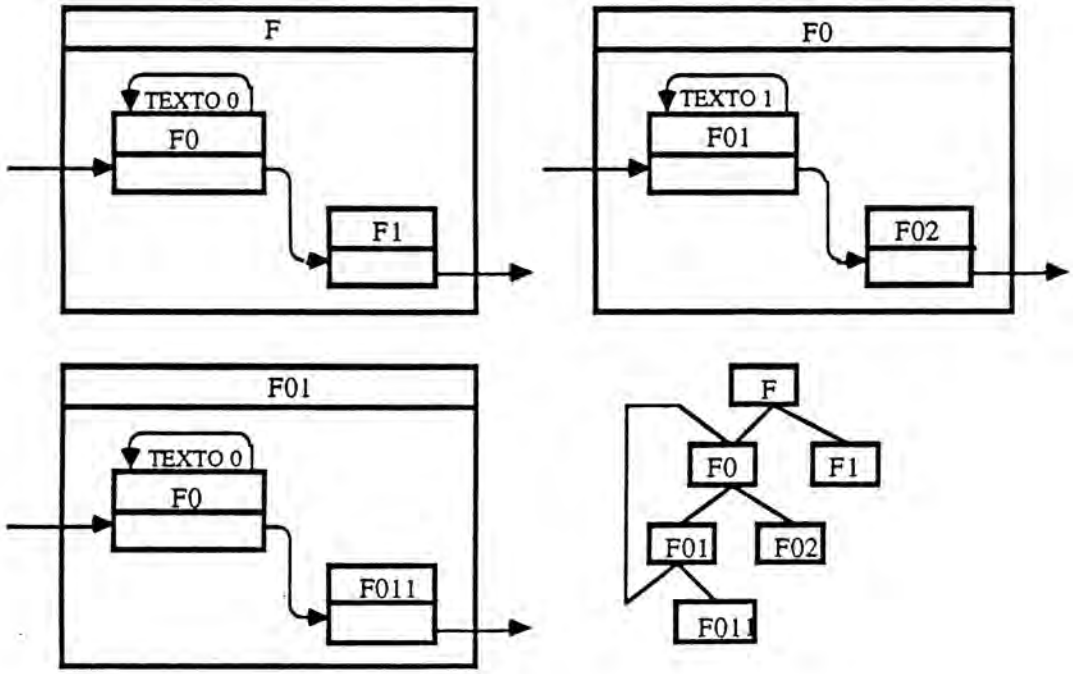


figura -5.13-

Diagramas ANA y representación en árbol de un caso de recursión (F0) que aparece en el segundo nivel de análisis.

Una función recursiva se representa mediante una línea de retorno en la parte superior de la caja correspondiente a la función y un texto auxiliar que puede aclarar el comportamiento o la finalidad de la recursión. En la figura -5.13- aparece una recursión en el segundo nivel de análisis de la función F0:obsérvese que tanto F0 como F01 son funciones recursivas.

5.4- Observaciones

Todo lo que está descrito aquí corresponde a la representación de los diagramas ANA de la primera versión de la herramienta. En la segunda habrá algunos cambios sobre todo referentes a la representación de decisiones (bloques IF): cada bloque de estos se representará mediante una función privada que se puede desarrollar interiormente en el mismo diagrama.

Además se esta desarrollando una serie de reglas de representación de algoritmos en diagramas ANA orientados al lenguaje de programación C. Con ello se quiere implantar un modelo estándar de lenguaje ANA, de la misma forma que el Standard C. Esto tiene grandes ventajas, no solo de comprensión, sino también de cara al futuro para futuras aplicaciones de la herramienta ANA, como se verá en el capítulo 9.

Capítulo 6

DISEÑO VERSIÓN 1

Al comenzar el desarrollo de esta versión, ya se contaba con un programa que representaba en pantalla un dibujo básico ANA. Este programa leía unos datos de un fichero ASCII y dibujaba en pantalla un diagrama ANA simplificado: la diferencia más importante con el desarrollo posterior de la herramienta radicaba en la representación de los bloques IF, que a la postre se verificó ser lo más complicado de todo el diagrama.

6.1- Desarrollo básico

6.1.1- Añadidos

Como ya hemos comentado en el apartado anterior, la diferencia más importante estaba en la representación de los bloques IF. Estos, en el dibujo base, estaban limitados a elegir entre funciones aisladas. Es decir, no permitía desarrollar opciones entre grupos de funciones. Hubo que modificar la representación de los bloques IF en los diagramas ANA para poder incluir esta opción básica de todo programador. Para ello se ideó incorporar un rectángulo adicional que englobase las funciones de cada opción del bloque IF por separado. Aquí se respetaría la

representación ANA de estos bloques, manteniendolos uno debajo de otro.

Otros cambios fueron una mejora de la política de cruce de líneas de datos. Muchos casos en donde se pueden evitar cruces no son evidentes de ver y muchos de ellos fueron descubiertos únicamente después de trabajar mucho con la herramienta y ver muchos casos de diagramas ANA. De ello se puede deducir que la solución a este problema no es en ningún caso sencilla dadas las numerosas posibilidades que hay. La solución no solo depende de por donde circulan las líneas sino sobre todo de como están ordenadas las entradas y salidas de los datos. Se decidió que la mejor manera de resolver este problema era hacer comparaciones individuales entre todos los destinos de las salidas para ver con que orden había menos cruces de líneas.

Otro problema que se tuvo que resolver fue el aprovechamiento del espacio en el diagrama ANA. Estos pueden llegar a ser bastantes extensos y la mayoría de ellos ocuparán más de una pantalla. Por ello era importante evitar espacios vacíos. La política de aprovechamiento de los espacios ha sido bastante complicada de implantar, sobre todo por los numerosos casos que se podían plantear.

También hubo que modificar los datos de los que se partía para dibujar los diagramas ANA ya que si éstos eran suficientes para poder

representar los diagramas, eran insuficientes a la hora de hacer modificaciones y mantener la coherencia del dibujo.

La última modificación de importancia que hubo que realizar fue la incorporación de nuevas estructuras intermedias entre los datos base y el dibujo definitivo en pantalla. Pero la descripción de estas estructuras tiene lugar en el siguiente apartado.

6.1.2- Estructuras

Entre los datos almacenados en el fichero y la representación definitiva en pantalla del diagrama ANA se emplean cuatro niveles de estructuras de datos.

La primera estructura ('lectur') es la que se rellena con los datos del fichero. Aquí están almacenados todos los datos necesarios en alto nivel. Con alto nivel se quiere decir que lo que esta almacenado son las relaciones entre los elementos del dibujo, siendo estos elementos funciones, datos, bucles, bloques IF, etc.. . En ningún caso se emplean de coordenadas de ningún tipo. Solo existen posiciones relativas de alto nivel entre las funciones que en esta estructura se denominan 'cajitas'. A continuación podemos ver la estructura 'lectur' al completo.

```
/*-------------------------------------------------------------------*/
/*                ESTRUCTURA LECTUR                                  */
/*-------------------------------------------------------------------*/

struct entrad {                 /* Entrada codificada de fichero */
                int             orden  ;
                int             usada  ;
                char            *nam   ;
                int             ndest  ;
                char            *(*nom);
                char            *(*des) ;
                char            *tipo   ; } ;

struct salida {                 /* Salida codificada de fichero */
                int             orden  ;
                int             usada  ;
                char            *nam   ;
                int             norig  ;
                char            *(*nom);
                char            *(*ori)  ;
                char            *tipo   ; } ;

struct cajita {                 /* Caja codificada de fichero */
                char            *id     ;
                int             ncadna ;
                char            *(*name);
                int             pohor  ;
                int             pover  ;
                int             nentra ;
                struct salida   *ent    ;
                int             nsalid ;
                struct entrad   *sal    ; } ;

struct condic {                 /* Dato tipo condicion codificado */
                int             norig  ;
                char            *(*nom);
                char            *(*ori)  ;
                char            *tipo   ; } ;

struct macro  {                 /* macro codificada de fichero */
                int             hueh1  ;
                char            *con   ;
                int             nivcon ;
                int             ncaj   ;
                char            *(*caj)  ; } ;

struct blif   {                         /* Bloque if codificado de fichero */
                int             hueh1  ;
```

```
                int             nivani  ;
                int             nmac    ;
                struct macro    *mac    ;
                struct condic   oc      ;
                int             bifpa   ;
                int             macma   ; } ;

struct bucli {                          /* Bucle codificado de fichero */
                int             ncaj    ;
                char            *(*caj) ;
                char            *texaux ;
                int             nivani  ;
                char            *texcon ;
                int             nivcon  ;
                char            tipo    ;
                char            *excon  ;
                struct condic   oc      ;
                struct condic   ob      ;
                struct condic   eb      ; } ;

struct cursi {                          /* Dato tipo recursion codificado */
                char            *caja   ;
                char            *tex    ; } ;

struct inici {                          /* Dato tipo recursion codificado */
                char            *caja   ;
                char            *inp    ;
                char            *val    ;
                char            *anca   ; } ;

struct lectur {                 /* Estructura auxiliar de lectura */
                struct cajita   tree    ;
                char            *id     ;
                char            *name   ;
                int             nentra  ;
                struct entrad   *ent    ;
                int             nsalid  ;
                struct salida   *sal    ;
                int             ncajis  ;
                struct cajita   *cs     ;
                int             ncajic  ;
                struct cajita   *cc     ;
                int             nblif   ;
                struct blif     *bif    ;
                int             nbuc    ;
                struct bucli    *buc    ;
                int             nrecur  ;
                struct cursi    *rec    ;
```

```
    int             ninic   ;
    struct inici    *ini    ; } ;
```

La siguiente estructura que tenemos se llama corfil. En un principio iba a ser la de más alto nivel pero luego resultó que ello no era posible. Ahora se ha convertido en una estructura intermedia entre 'lectur' y la siguiente de más bajo nivel. Aquí se empiezan a almacenar algunas coordenadas de algunos elementos pero siempre en alto nivel. Esta estructura se ha convertido en un suplemento de la siguiente estructura, llamada 'dibujo', que en la siguiente versión desaparecerá. A continuación se representa el contenido de la estructura 'corfil'.

```
/*-------------------------------------------------------------------------*/
/*                ESTRUCTURA CORFIL                */
/*-------------------------------------------------------------------------*/

struct coohor {                 /* Coord. horizontales de un punto */
                int             sr      ;
                int             hu      ;
                int             tw      ; } ;

struct coover {                 /* Coord. verticales de un punto */
                int             sr      ;
                int             th      ;
                int             vu      ; } ;
struct inpout {                 /* Un dato input o output */
                char            *s      ;
                struct coohor   ch      ;
                struct coover   cv      ; } ;

struct caja   {                 /* Una caja */
                char            *id     ;
                int             pohor   ;
                int             pover   ;
                int             ncadna ;
```

```
                char            *(*name);
                int             ninp    ;
                struct inpout   *inp    ;
                int             nout    ;
                struct inpout   *out    ; } ;

struct fledat {                 /* Una flecha de tipo dato */
                int             norig   ;
                struct inpout   *(*or)  ;
                int             ndest   ;
                struct inpout   *(*de)  ; } ;

struct bloqif {                 /* Un bloque if */
                int             hueh0   ;
                int             hueh1   ;
                int             nivani  ;
                int             nmac    ;
                int             *huev0  ;
                int             *huev1  ;
                int             *nivcon ;
                int             *ncaj   ;
                char            *(*con) ;
                int             norcon  ;
                struct inpout   *(*oc)  ; } ;

struct bucle  {                 /* Un bucle */
                int             hueh0   ;
                int             hueh1   ;
                int             huev0   ;
                int             huev1   ;
                char            *texaux ;
                int             nivani  ;
                char            *texcon ;
                int             nivcon  ;
                struct inpout   extcon  ;
                char            tipo    ;
                int             norcon  ;
                struct inpout   *(*oc)  ;
                int             norbu   ;
                struct inpout   *(*ob)  ;
                int             nexbu   ;
                struct inpout   *(*eb)  ; } ;

struct inicia {                 /* Una inicializacion */
                char            *valor  ;
                struct inpout   *var    ;
                int             hue     ; } ;
struct recurs {                 /* Una recursion */
```

```
                int             pohor   ;
                int             pover   ;
                char            *texcon ; } ;

struct corfil {                 /* Informacion de fichero */
                char            *id     ;
                char            *name   ;
                int             ninp    ;
                struct inpout   *inp    ;
                int             nout    ;
                struct inpout   *out    ;
                int             ncajs   ;
                struct caja     *cs     ;
                int             ncajc   ;
                struct caja     *cc     ;
                int             nbloif  ;
                struct bloqif   *bi     ;
                int             nbuc    ;
                struct bucle    *bu     ;
                int             nrecur  ;
                struct recurs   *re     ;
                int             ninic   ;
                struct inicia   *in     ;
                int             nfled   ;
                struct fledat   *fd     ; } ;
```

La tercera estructura es la llamada 'dibujo'. En esta estructura ya los elementos dejan de ser de alto nivel: ya no hay funciones ni datos ni elementos similares. Ahora tenemos líneas, rectángulos, texto, etc.. . Estos elementos están relacionados entre sí con coordenadas de alto nivel. Esto quiere decir que en el dibujo existen varios sistemas de referencia en los cuales los elementos están situados en números de unidades de ancho o alto de letra y números de unidades horizontales y verticales. Con ello tenemos representado el dibujo completo sin una escala dada. Esto será muy útil para cuando se quiera modificar la escala de la representación del diagrama ANA en pantalla: no hará falta

reconstruir la estructura 'dibujo' y por lo tanto tampoco la estructura 'corfil'. A continuación se incluye la estructura 'dibujo'.

```
/*-------------------------------------------------------------------*/
/*                ESTRUCTURA DIBUJO                                  */
/*-------------------------------------------------------------------*/

struct unidad {                 /* Magnitudes unitarias del dibujo */
                int             tw      ;
                int             hu      ;
                int             th      ;
                int             vu      ; } ;

struct cajhor {                 /* Datos de las cajas horizontales */
                int             nchi    ;
                int             ncho    ;
                int             nchn    ; } ;

struct cajver {                 /* Datos de las cajas verticales */
                int             ncad    ;
                int             nvu     ; } ;

struct dimcaj {                 /* Dimensiones de las cajas */
                int             ncah    ;
                struct cajhor   *cah    ;
                int             ncav    ;
                struct cajver   *cav    ; } ;

struct dimhue {                 /* Dimensiones de los huecos */
                int             nchmar;
                int             nhuh    ;
                int             *ntw    ;
                int             *nhu    ;
                int             nhuv    ;
                int             *nth    ;
                int             *nvu    ; } ;

struct coohor {                 /* Coord. horizontales de un punto */
                int             sr      ;
                int             hu      ;
                int             tw      ; } ;

struct coover {                 /* Coord. verticales de un punto */
```

```
                int             sr      ;
                int             th      ;
                int             vu      ; } ;

struct texele {                 /* Texto elemental */
                struct coohor   ch      ;
                struct coover   cv      ;
                char            *s      ; } ;

struct texdib {                 /* Textos de un dibujo */
                int             ntdat   ;
                struct texele   *td     ;
                int             ntcaj   ;
                struct texele   *tc     ;
                struct texele   tt      ; } ;

struct rectan {                 /* Rectangulo simple */
                struct coohor   ch      ;
                struct coover   cv      ;
                struct unidad   dr      ; } ;

struct arcrec {                 /* Arco de angulo recto */
                struct coohor   *ch     ;
                struct coover   *cv     ;
                int             cuadr   ; } ;

struct linfin {                 /* Una linea finas */
                struct coohor   *x0     ;
                int             rx0     ;
                struct coover   *y0     ;
                int             ry0     ;
                struct coohor   *x1     ;
                int             rx1     ;
                struct coover   *y1     ;
                int             ry1     ;
                int             modx[2]         ;
                int             mody[2]         ; } ;

struct lingru {                 /* Una linea gruesa */
                struct coohor   ch[2]   ;
                struct coover   cv[2]   ; } ;

struct lineas {                 /* Conjunto de lineas */
                int             nlinf   ;
                struct linfin   *lf     ;
                int             nling   ;
                struct lingru   *lg     ; } ;
```

```
struct flecha {                 /* Una punta de flecha */
        struct coohor   *ch     ;
        int             rch     ;
        struct coover   *cv     ;
        int             rcv     ;
        int             npimed ; } ;

struct marca1 {                 /* Una marca en un bucle */
        struct coohor   *(coh[2]);
        struct coover   *cov    ;
        char            num[3] ; } ;

struct marca2 {                 /* Una marca en una macro */
        struct coohor   coh     ;
        struct coover   cov     ;
        char            num[3] ; } ;

struct relele {                 /* Relacion de elementos de dibujo */
        struct texdib   te      ;
        int             nrect   ;
        struct rectan   *re     ;
        int             narc    ;
        struct arcrec   *ar     ;
        struct lineas   li      ;
        int             nfle    ;
        struct flecha   *fl     ;
        int             nmar1   ;
        struct marca1   *ma1    ;
        int             nmar2   ;
        struct marca2   *ma2    ; } ;

struct dibujo {                 /* Un dibujo en formato */
        struct unidad   dr      ;
        struct dimcaj   dc      ;
        struct dimhue   dh      ;
        struct relele   re      ; } ;
```

Por último aparece la estructura 'grafic'. Esta última estructura es la de más bajo nivel y la representación de los mismos elementos que en dibujo pero ya concretados en una escala dada y pasados a coordenadas absolutas en píxels. Con esta estructura ya solo queda utilizar las

funciones gráficas del compilador utilizado para programar (En nuestro caso las funciones gráficas del Turbo C). Esta estructura es bastante útil cuando se desea moverse por un dibujo que ocupa más de una pantalla: al estar definidos todos los elementos en coordenadas de píxels, sólo hay que sumarle una cantidad constante para que lo que aparezca en la pantalla sea lo que se desea. No hay que volver a calcular ninguna coordenada ni por lo tanto modificar ninguna estructura anterior. A continuación se presenta la estructura 'grafic'.

```
/*-------------------------------------------------------------------------*/
/*                    ESTRUCTURA GRAFIC                    */
/*-------------------------------------------------------------------------*/

struct rect     {           /* rectangulo en píxels */
                int         pih     ;
                int         piv     ;
                int         pfh     ;
                int         pfv     ; } ;

struct line     {           /* linea */
                int         pih     ;
                int         piv     ;
                int         pfh     ;
                int         pfv     ; } ;

struct teg      {           /* texto en píxels */
                int         x       ;
                int         y       ;
                char        *s      ; } ;

struct text     {           /* conjunto de textos */
                struct teg  tgt     ;
                int         ntcj    ;
                struct teg  *tgc    ;
                int         ntd     ;
                struct teg  *tgd    ; } ;
```

```
struct fle      {           /* flecha */
                int         x[6]    ; } ;

struct arc      {           /* arco */
                int         x       ;
                int         y       ;
                int         iz      ;
                int         fz      ;
                int         ra      ; } ;

struct marc     {           /* marca */
                int         x[8]    ;
                struct teg  tgm     ; } ;

struct grafic   {           /* elementos del dibujo */
                int         nrect   ;
                struct rect *re     ;
                int         nline   ;
                struct line *li     ;
                struct text tx      ;
                int         nfle    ;
                struct fle  *fl     ;
                int         narc    ;
                struct arc  *ar     ;
                int         nmar    ;
                struct marc *ma     ; } ;
```

Hay por tanto cuatro niveles de definición del diagrama ANA. Los tres primeros tuvieron que ser adaptados a las nuevas necesidades y el último fue creado para esta versión. Como ya se ha dicho cada uno tiene su utilidad y sobre todo son niveles a los que se pueden acceder para diversas actuaciones. Esto esta ilustrado en la figura -6.1- .

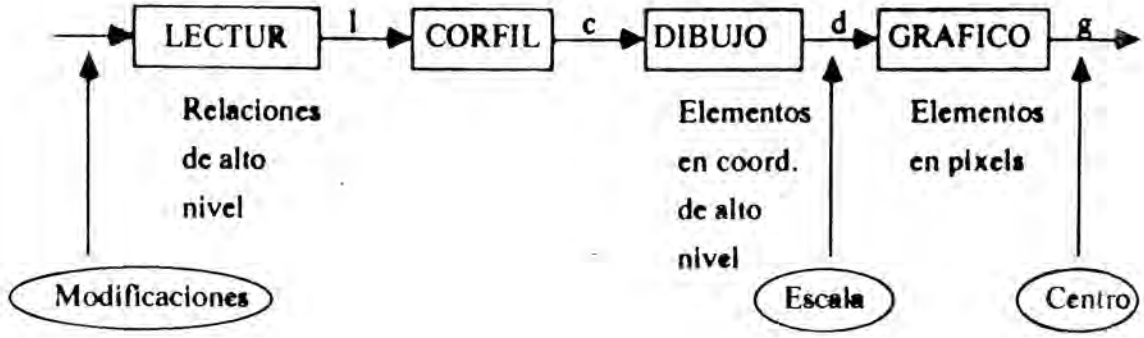


figura -6.1-

Descripción de las estructuras y su utilidad

6.2- Menús

6.2.1- Aparición

Para poder realizar una herramienta realmente fácil de usar se pensó en sistemas de menús para poder acceder a todas las opciones que se iban a implantar. Estos menús debían ser de alto nivel y fáciles de utilizar. Por ello se eligió los menús encadenados: cuando se elige una opción que tiene a su vez varias posibilidades aparece superpuesto otro menú y así sucesivamente.

Estos menús debían ser agradables a la vista puesto que se iba a trabajar constantemente con ellos. Por ello se hicieron con todo tipo de colores que como ya se verá posteriormente se pueden configurar al gusto de cada uno.

Existen cuatro tipos de menús básicos diferentes en la herramienta. El primero de ellos es el más normal: un menú donde existen una serie de opciones fijas. Se puede acceder a ellas colocandose encima y pulsando la tecla 'ENTER' o invocando la letra clave que aparece de otro color. Estos son los más numerosos. Se puede ver un ejemplo de estos en la figura -6.2- donde vemos el menú principal de la herramienta.

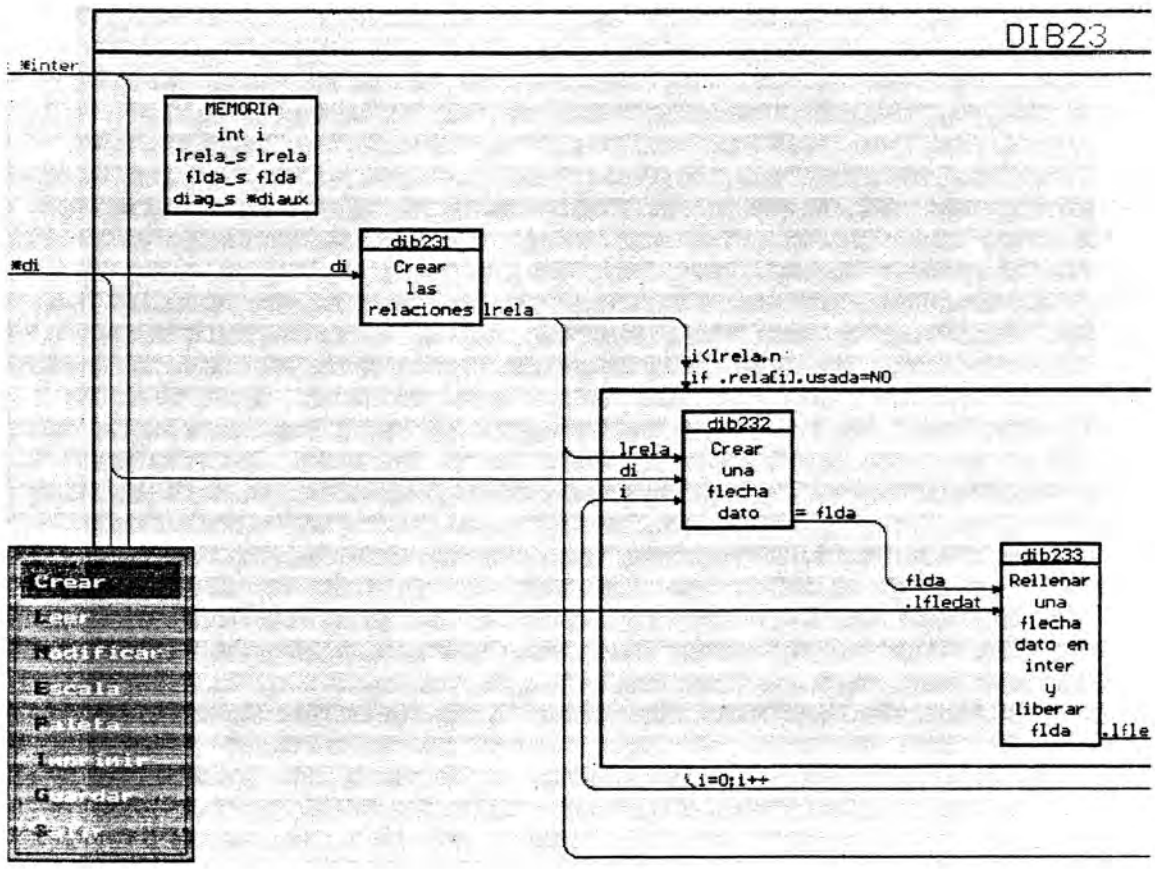


figura -6.2-

Menú de opciones fijas

Otro tipo de menú es el variable: hay que elegir un elemento que depende del estado del dibujo. Por ejemplo, se quiere modificar una función dada y se tiene que elegir entre las que en ese momento hay en el diagrama ANA de la pantalla. En estos menús, por razones obvias, no existe la posibilidad de acceder a las opciones mediante la letra clave. Sólo se puede elegir mediante las flechas del teclado y la tecla 'ENTER'. No se contempla la utilización del ratón en nigún momento. Además estos menús aparecen con un título explicativo de lo que son. Otro problema que hubo que resolver con estos menús fue el número de opciones que podían tener. Al ser variables, podía haber desde una opción hasta un número limitado únicamente por la memoria del ordenador. Por lo tanto sólo se permite ver un máximo de ocho opciones a la vez y se accede al resto mediante un "scroll" del menú con las teclas 'AvPág' y 'RePág'. Podemos ver un ejemplo de ello en la figura -6.3- donde aparece el menú de posibles ficheros para leer y cargar en pantalla.

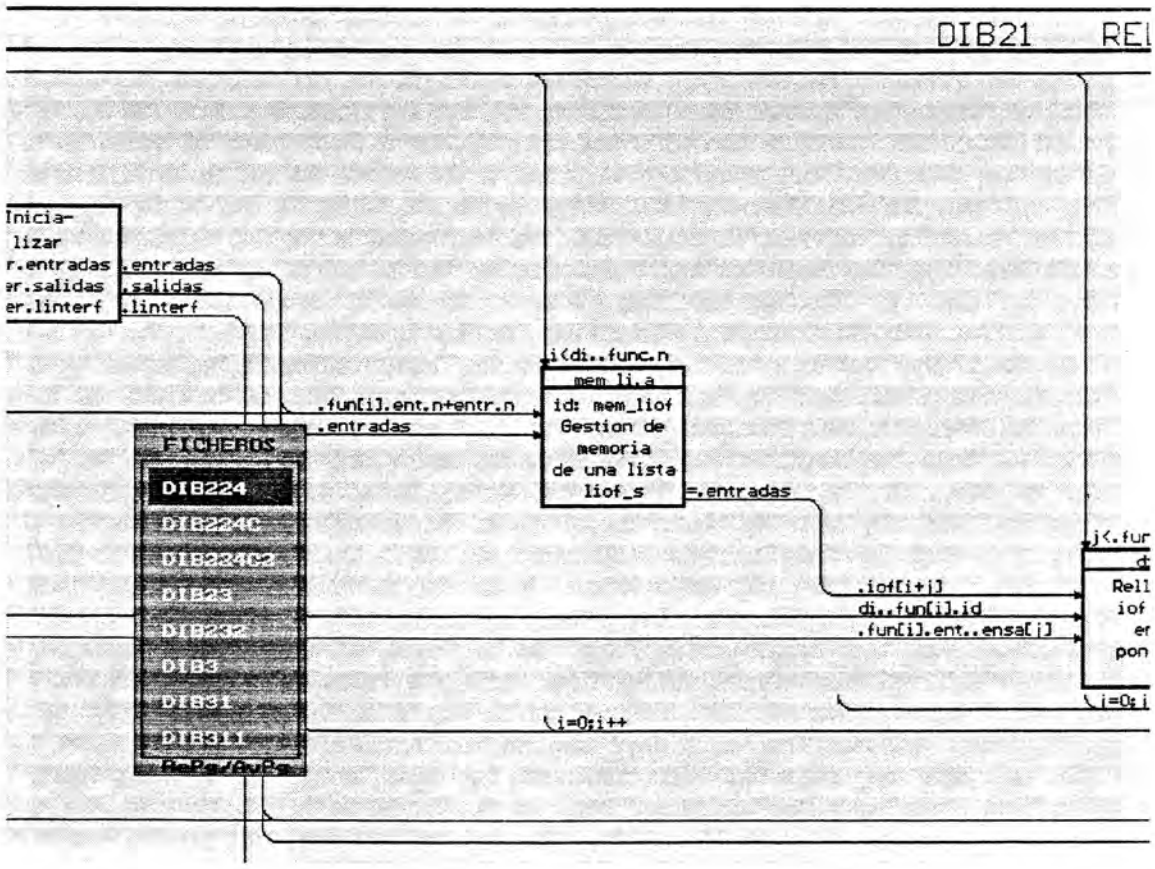


figura -6.3-

Menú variable con más de ocho opciones

Un tercer tipo de menú es el aceptación de datos: aquí tenemos varios subtipos. A lo largo del programa se pueden aceptar distintos elementos: hileras de caracteres, enteros y reales. Cada uno tiene su pequeña particularidad, incluso los del mismo tipo. El editor utilizado es el más sencillo: solo se puede borrar lo que se ha escrito y volver a

escribirlo; no permite modificaciones. En la figura -6.4- podemos ver el menú de entrada del nombre de una función. Ya se han introducido varias líneas.

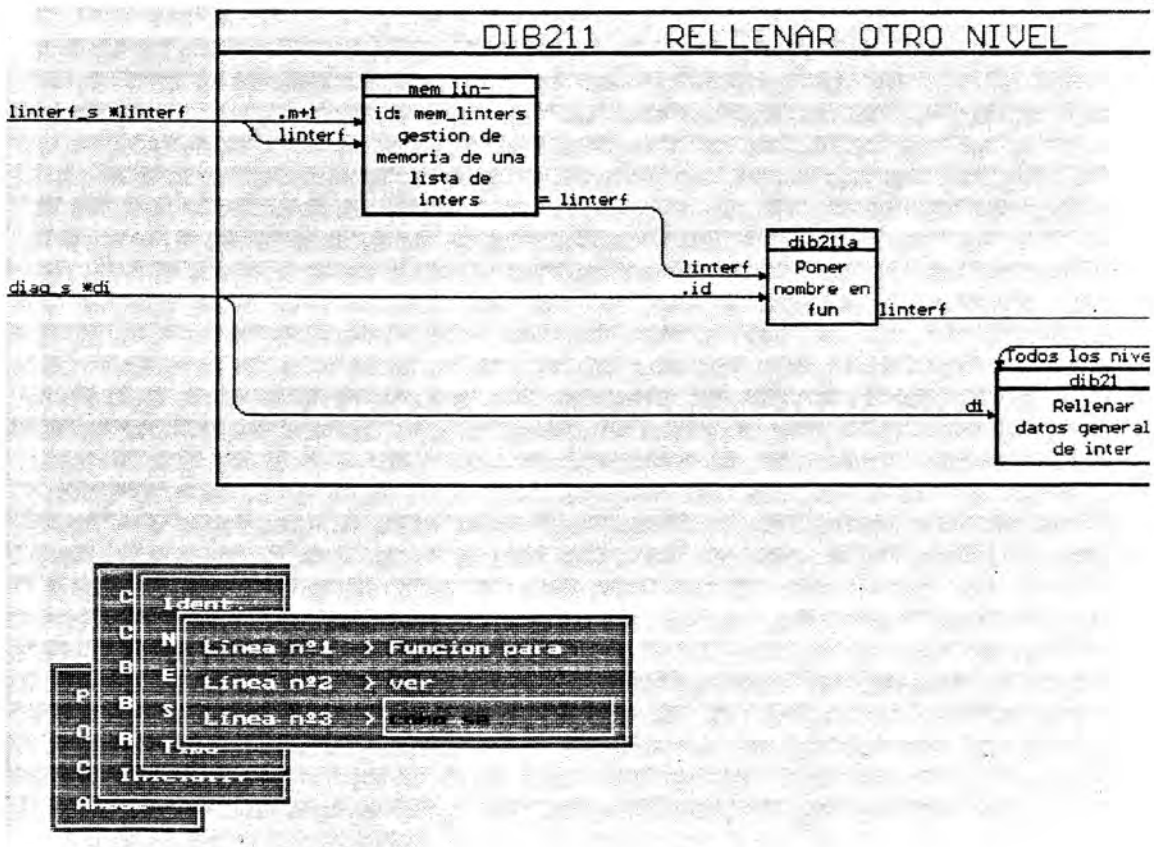


figura -6.4-

Menú de entrada de datos

Y por último tenemos los menús que no tienen ninguna opción. Sencillamente son de aviso o las opciones son SI/NO. La manera de manejarlos viene en el menú mismo, siendo lo más común que desaparezcan al pulsar cualquier tecla. Tenemos un ejemplo de ello en la figura -6.5-.

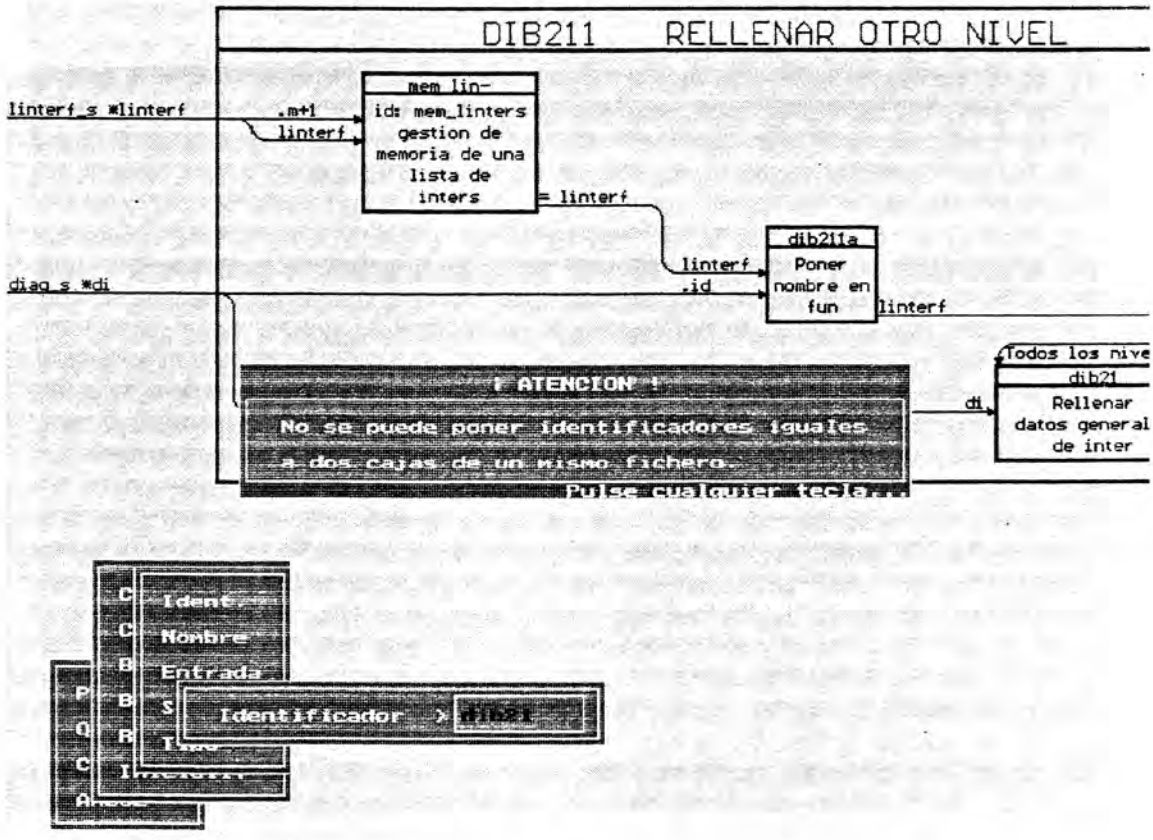


figura -6.5-

Menú de aviso

6.2.2- Problemas

El problema más importante que existe al diseñar los menús fue la cantidad de memoria que hacía falta para poder almacenar lo que estos escondían y que había que poder recuperar posteriormente. Para resolver esto se almacenan cada menú (en realidad lo que tapa) en un fichero temporal que llevamos al disco duro. De esta manera solo teníamos a la vez un menú en memoria. Aún así cuando se maneja un diagrama grande sigue habiendo problemas. En estos casos, en vez de interrumpir la ejecución del programa, no se almacenan lo que hay debajo del menú y sencillamente se rellenan el recuadro que deja con color de fondo. Esta política va a cambiar en la segunda versión ya que no es muy alentador ver como desaparece parte de la pantalla.

6.2.3- Política de menús

Cuando se habla de política de menús nos referimos a qué acciones desencadenan que hechos. Ya sabemos que en un menú se puede elegir entre varias opciones colocandose sobre ella y pulsando la tecla 'ENTER' o tecleando la letra clave. Pero existen otras acciones.

Primero, para salir de cualquier menú en el que nos encontremos basta con pulsar la tecla 'ESC'. De esta manera volvemos al menú anterior o al estado de ningún menú, según en que sitio se esté.

A lo largo de la utilización de la herramienta se modifican cosas en el diagrama ANA en el que se esta trabajando. Estas modificaciones a veces implican varios pasos. La política que se sigue en estos casos es que cualquier modificación que tenga sentido y ya este acabada se incorpora al dibujo. Si antes de que esto ocurra se pulsa la tecla 'ESC', se da por finalizada y se cancela la acción de modificar.

Otra política que se sigue es que cuando en vez de que se presenten varias opciones posibles solo haya una, no se pregunta al usuario y se prosigue en esa opción. Esto solo tiene una excepción y es cuando se esta quitando algo: en estos casos se sigue un criterio de prudencia y se pregunta antes de proceder.

6.3- Mantenimiento de 'lectur'

6.3.1- Política de permisividad

Lo primero que se planteó cuando se empezó a diseñar esta parte del programa fue qué se permitiría hacer al usuario y qué no. La respuesta nos pareció clara: solo se le permitiría añadir elementos coherentes en el dibujo. La razón de ello es que de esta manera se le evitan errores al usuario (la herramienta funcionaría a modo de compilador en este caso) y nos permitiría mantener un dibujo siempre lógico y por lo tanto infinitamente más fácil de manejar.

6.3.2- Problemas

Cuando se empezó a implantar el mantenimiento de lectur, aparecieron toda una serie de problemas que se fueron resolviendo poco a poco salvo uno: modificar elementos que estaban dentro de bloques IF. Hay que recordar que la primera representación de diagramas ANA de la cual se partió para la realización de este programa no contemplaba estos nuevos bloques IF donde se describen varias funciones en cada opción.

Se intentaron resolver los problemas que se planteaban pero al desaparecer uno aparecían dos insospechados. Tal como esta la herramienta no permite hacer todos los cambios posibles. Fue en gran parte debido a esto que surgió la posibilidad de la segunda versión que contemplase desde un principio estos problemas en vez de poner parches en la primera versión. Había que modificar cualitativamente la base del diagrama.

6.3.3- Árbol de menús

En este apartado del mantenimiento de 'lectur' o, dicho con otras palabras, de modificación del diagrama en pantalla (i.e. en memoria) es donde el árbol de menús es más amplio. Parte de la opción 'Modificar' del menú principal. El primer menú que aparece (Ver figura -6.6-) tiene cuatro opciones básicas. La más extraña es la última: 'Árbol'. Esta es para poder cambiar los datos de la función que estamos describiendo: es

decir los datos de la función que es el diagrama ANA en cuestión. Las otras no presentan ninguna dificultad son las clásicas de modificar elementos.

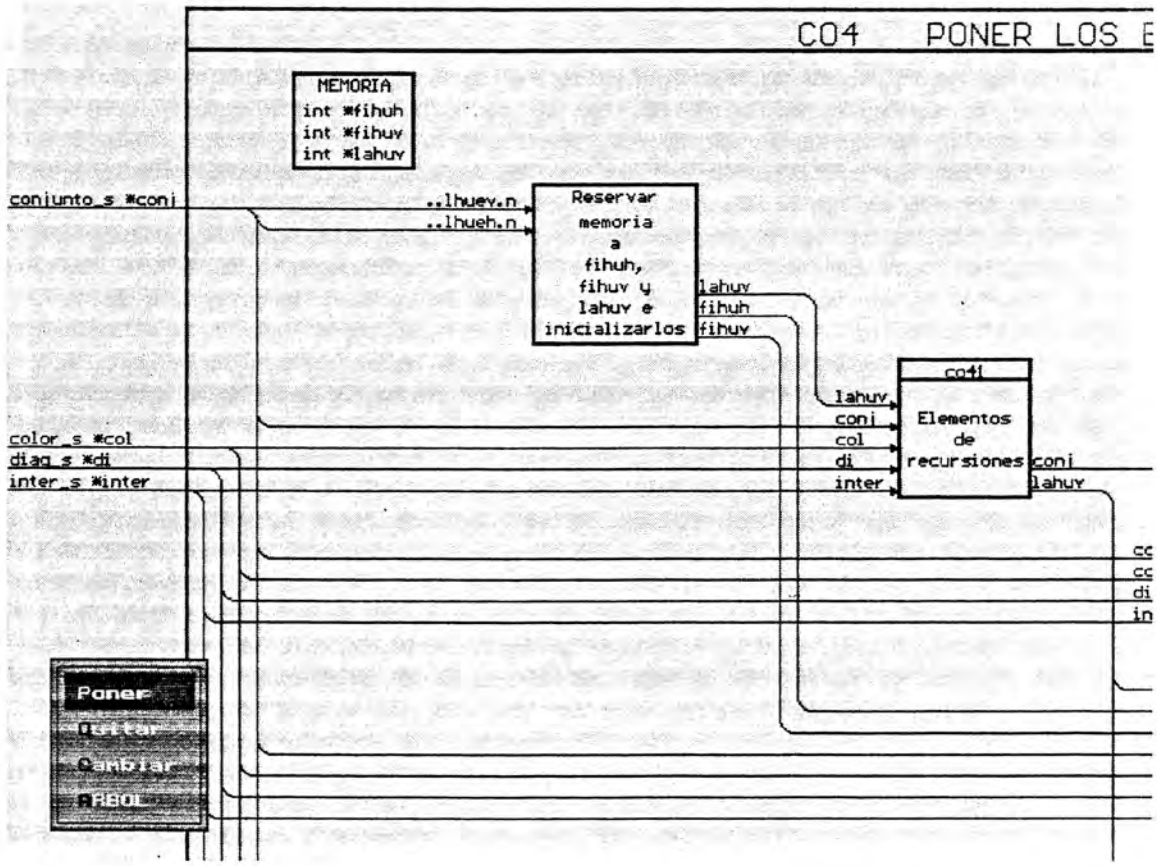


figura -6.6-

Menú de la opción 'Modificar'

En realidad la tercera es redundante ya que cambiar algo es al fin y al cabo lo mismo que quitar y poner, y de hecho todo lo que hace el menú 'Cambiar' podría hacerse con los otros si no fuese por los bloques IF. Pero resultó ser más cómodo para el usuario y también más claro. En cualquier caso se puede ver en la figura -6.7- en que aparece el menú de la opción 'Poner' que se accede directamente a todos los elementos que hay en un dibujo. La opción 'Quitar' tiene un menú idéntico a este y 'Cambiar' también con la salvedad de los datos que no se cambian: solo se quitan y se ponen, debido a la propia naturaleza de los flujos de datos, que no tienen entidad propia como elementos.

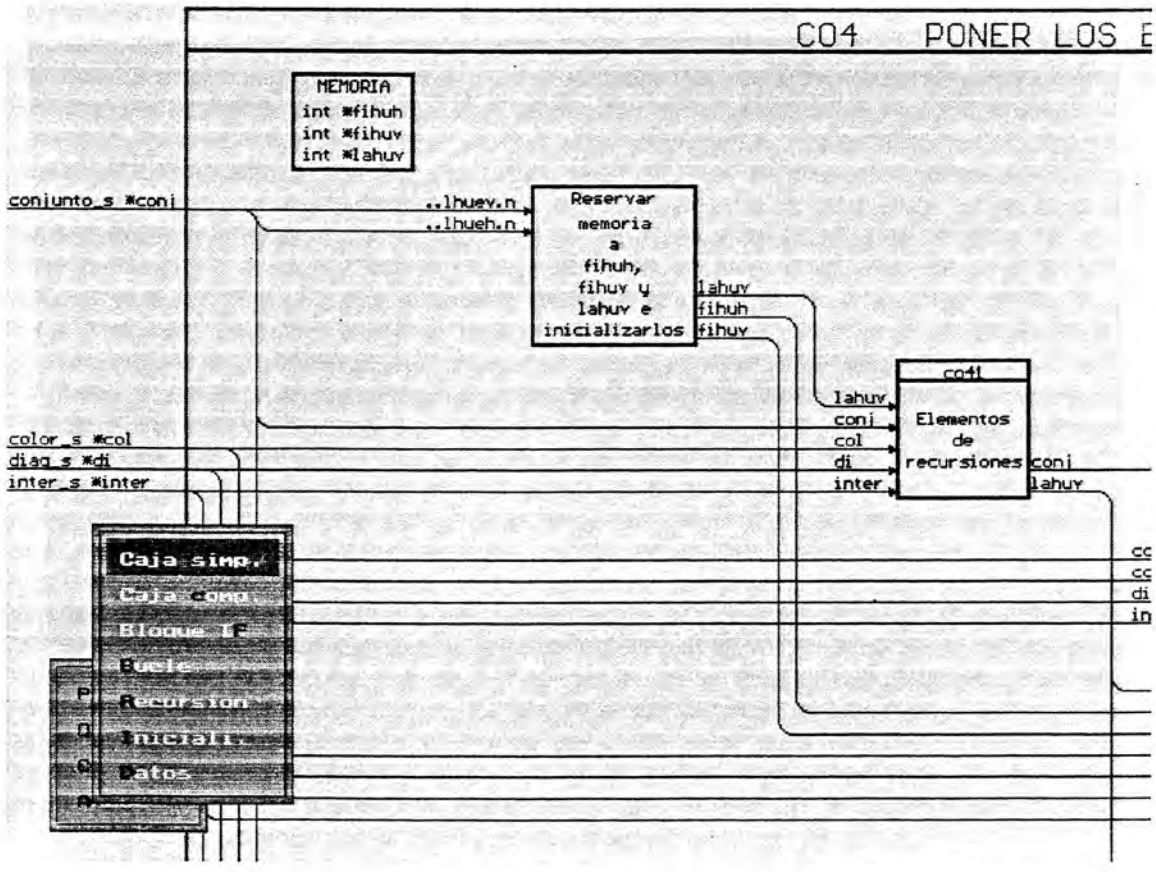


figura -6.7-

Menú de la opción 'Poner'

6.4- Escalas

Cuando tenemos un dibujo en pantalla podemos querer verlo en distintos tamaños para poder tener una visión de conjunto o sencillamente ver un detalle. Para ello se ha desarrollado la posibilidad de distintas escalas de representación del dibujo en pantalla.

Internamente, como ya se ha comentado anteriormente, para llevar a cabo estos cambios, solo hay que modificar la estructura 'gráfico', partiendo de la ya creada 'dibujo'. Las opciones de que se disponen aparecen el menú 'Escala' (Ver figura -6.8-).

Tenemos varias posibilidades de escalas:

- La opción por defecto es tamaño automático. Aquí se conservan las relaciones de aspecto y el tamaño de los distintos elementos es independiente del tamaño total del dibujo.

- Podemos elegir ver el dibujo en números fijos de pantallas, con tamaños estándares DIN.

- Por último podemos elegir el número de pantallas que vaya a ocupar el dibujo, siendo independientes las horizontales y las verticales, y pudiendo elegir un número decimal de pantallas.

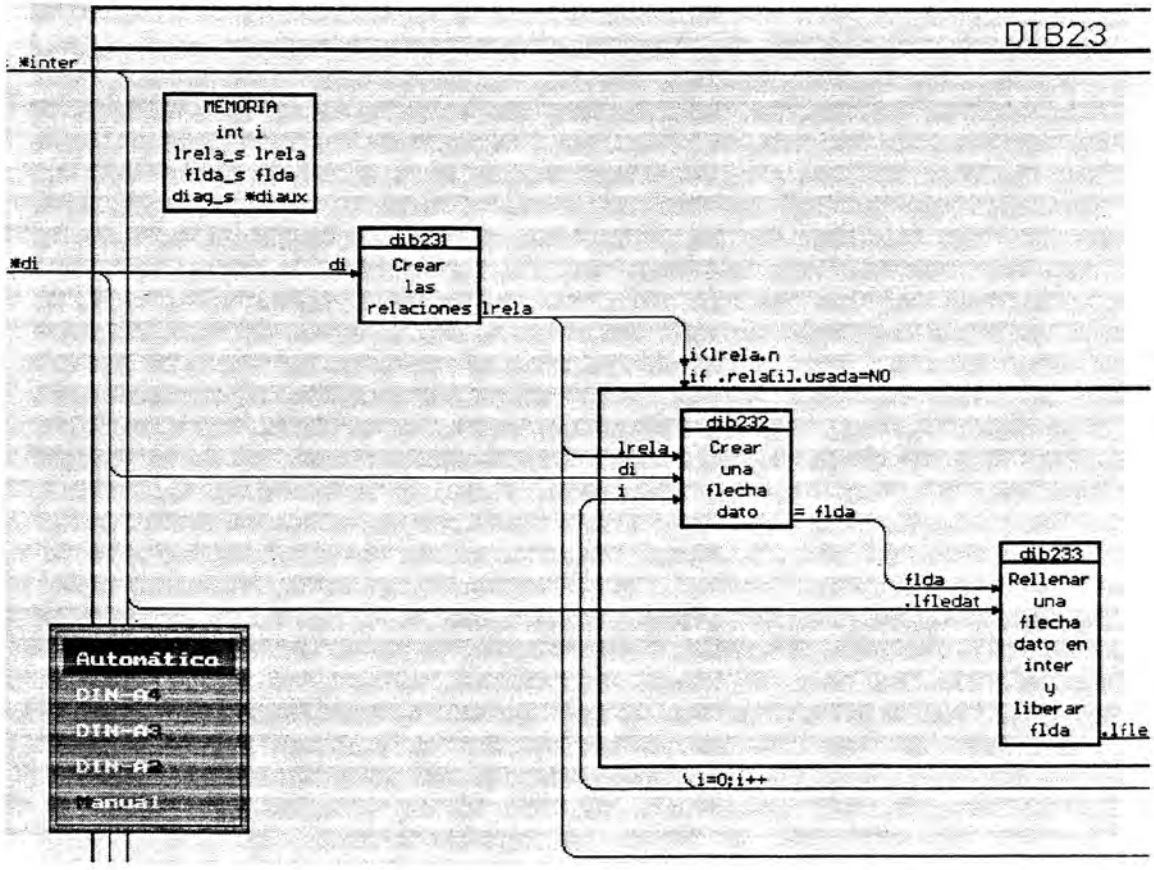


figura -6.8-

Menú de escalas

6.5- Navegación por un dibujo dado

Con esta opción se permite tener un diagrama ANA que ocupe más de una pantalla. Lo que se consigue es visualizar la parte del dibujo que se desee. Para poder navegar por el dibujo es condición necesaria que no halla ningún menú en pantalla. Esto se hizo para que no hubiese

problemas de rapidez en el movimiento de pantalla pero ha resultado ser bastante incómodo. En la siguiente versión se subsanará este problema.

Una de las grandes ventajas de esta opción es que, al tener el diagrama ya almacenado en píxels en la estructura 'gráfico', no hay que recalcular nada; solo hay que redibujarlo con otro origen.

Existen tres posibilidades de movimiento:

- Primero podemos movernos por pantallas completas, ya sean horizontales o verticales. Esta opción permite movimientos rápidos por el dibujo.

- También podemos movernos en diagonal media pantalla. Se eligió esta posibilidad dada la naturaleza del diagrama ANA.

- La última opción es moverse prácticamente de manera continua mediante un "marco" que se mueve a voluntad por el usuario, representando los bordes del futuro dibujo. Sirve para situarse en el sitio exacto que se desea.

6.6- Impresión

En esta versión se pueden imprimir los diagramas ANA. La opción de impresión que ofrece es muy práctica, pues no se requiere la conexión

física con una impresora sino que crea un fichero. Este fichero esta escrito en PostScript. La razón de la elección de este lenguaje es por dos razones: la primera es que nos permite hacer muchas cosas muy fácilmente al ser un lenguaje de alto nivel muy completo; la segunda es por que es un lenguaje altamente estandarizado: cualquier impresora con traductor PostScript será capaz de entender estos ficheros. De la misma manera se esperaba que se pudiese importar a cualquier programa de tratamiento de textos o de imagen. En esta versión no es así ya que le faltan ciertos requisitos de estandarización pero se subsanará en la siguiente versión. En cualquier caso este fichero, al ser ASCII, es igual de transportable que cualquier fichero de diagrama ANA.

Otra razón importante es que nos permite describir el diagrama con una precisión perfecta: con ello queremos decir que siempre utilizará las posibilidades de la impresora al máximo.

Una vez que se entra en el menú de impresión (ver figura -6.9-), tenemos una serie de opciones:

- Podemos imprimir (no olvidemos que cuando decimos imprimir queremos decir crear un fichero PostScipt) cualquier diagrama cuyo fichero este guardado en el directorio de trabajo o el diagrama que tenemos en memoria en ese momento. Se imprimir el fichero que aparece en el menú.

- De igual manera que para la representación en pantalla podemos elegir el tamaño del diagrama en la impresión (no olvidar que aquí tenemos mucha más libertad para tamaños pequeños ya que la impresora suele tener mayor definición). Si el tamaño es mayor de un DIN-A4, saldrá a modo de póster.

- Se puede elegir el tipo de letra (hay cuatro opciones distintas) aunque la más aconsejable sea 'COURIER' ya que es la que mejor aprovecha el espacio.

- Aparte del tamaño general del diagrama podemos modificar las relaciones entre las letras y los huecos mediante las opciones de número de puntos (tamaño de letra respecto al grosor de las líneas, según se quiera que resalten o no) y porcentaje de huecos verticales u horizontales (en proporción al tamaño de las letras).

Una vez todo listo para la impresión, se selecciona la opción 'O.K.' que crea el fichero tal como se le ha dicho.

En la figura -6.9-, tenemos el ejemplo de impresión del diagrama 'dib21' (notese que no es el que está en pantalla) en tamaño DIN-A4, con las letras de 9 puntos (Esto solo hará que se guarde la relación con las líneas ya que el tamaño de las letras se acoplará para que el diagrama quepa en un DIN-A4) y los huecos al cincuenta por ciento.

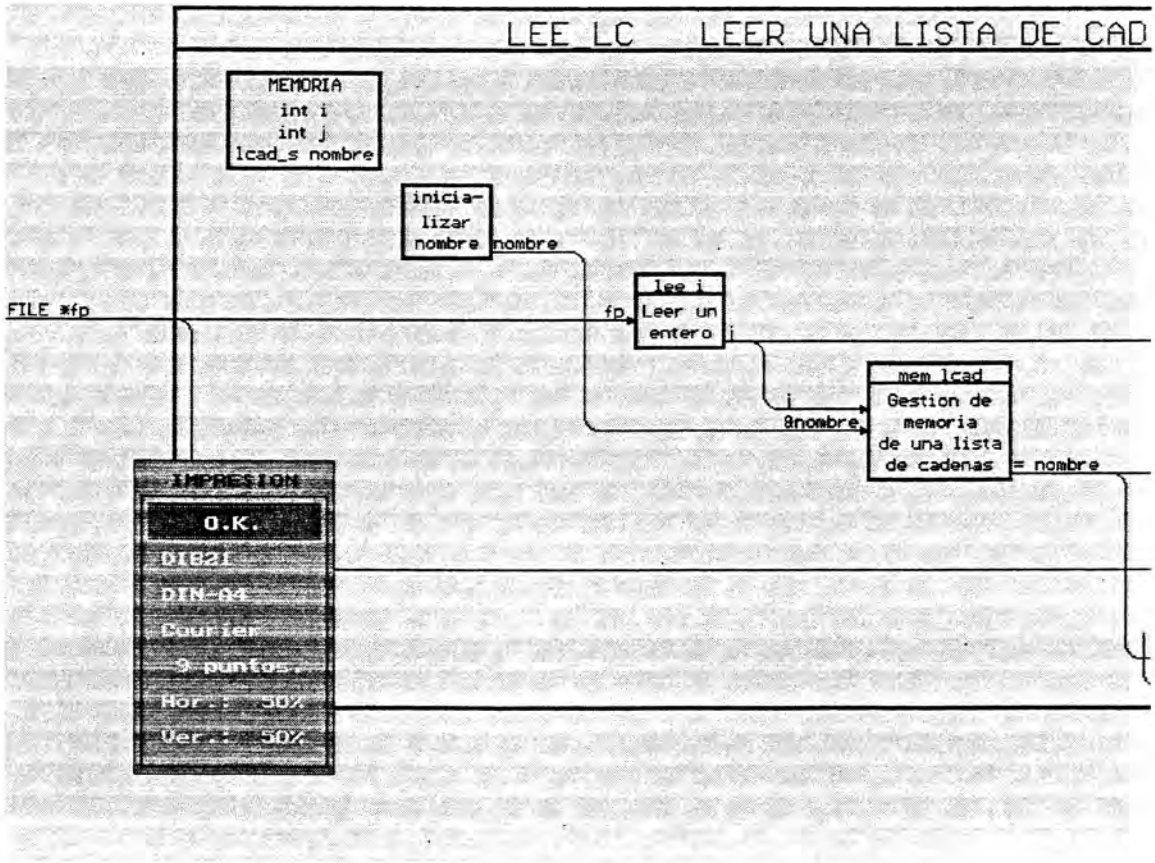


figura -6.9-

Menú de impresión

6.7- Configuración

En esta versión la configuración de la herramienta se limita a poder cambiar los colores de todos los elementos que aparecen en pantalla. El problema es el usuario no puede guardar su propia configuración para utilizaciones posteriores. La configuración que elija se mantiene solo

durante la sesión de trabajo. Una vez sale de la herramienta esa configuración se pierde.

La configuración de los colores la consigue mediante la opción 'PALETA' del menú principal. Una vez ahí se llega a un menú en el que se lista todos los elementos del dibujo. Al elegir uno aparece la paleta de colores (Ver figura -6.10-) en la que se selecciona el color deseado. Una vez hecho esto, se vuelve al estado 'sin menús en pantalla' con el cambio de color ya actualizado.

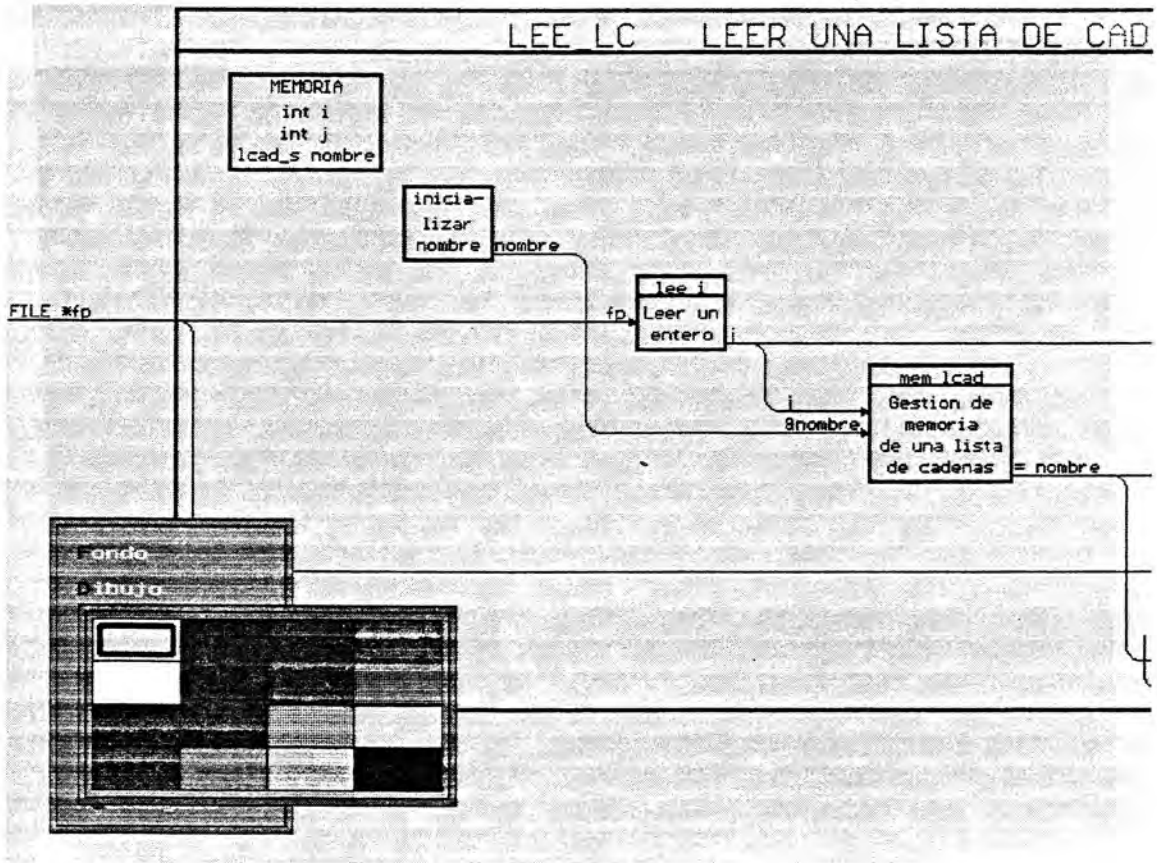


figura -6.10-

Pantalla para elegir color de paleta

6.8- Política de ficheros

Lo primero que se pensó fue la importancia de la transportabilidad de los diagramas ANA. Estos dibujos debían ser almacenados en ficheros lo más estándar posible de manera que fuesen fácilmente recuperables, transportables y manejables. Por ello se decidió que los

diagramas ANA se guardarían en ficheros ASCII. De esta manera se garantizaba su estandarización y su transportabilidad, incluso por correo electrónico si es necesario.

Una vez decidido esto había que decidir cuándo se guardarían estos ficheros. Lo que hace la herramienta es que si se ha entrado en los menús de modificar con un fichero abierto, siempre preguntará si se quieren guardar los cambios antes de cerrarlo. El fichero en el que se guarda la información se llamará con el mismo identificador de la función más la extensión 'ANA'. Por ejemplo, si el diagrama describe la función 'pepe', el fichero se llamará 'PEPE.ANA'.

En el caso de que ya se hubiese guardado anteriormente ese mismo diagrama, el antiguo fichero se renombra con la extensión 'BK' . De esta manera se conserva la anterior copia en el caso de que se halla guardado la nueva por error. La razón por la que no se renombra con la extensión 'BAK', mucho más común en estos casos es para que no se confunda, en el caso de escribir el código en el mismo directorio y se supone en un fichero con el mismo nombre del identificador pero con una extensión distinta de 'ANA', con la copia antigua de este. Por supuesto, el antiguo 'BK' se pierde.

La herramienta ANA trabaja con otro tipo de ficheros: los que genera la impresión. Estos ficheros también son ASCII y por lo tanto igual de transportables y manejables que los ficheros de diagramas. Lo

que contienen es código PostScript que cualquier impresora con traductor entiende e imprime. Estos ficheros se llamarán del mismo modo que los de diagramas pero con la extensión 'PS' (por ejemplo 'PEPE.PS').

En cuanto a dónde se guardan los ficheros o de dónde se leen estos ficheros, en esta versión es muy sencillo: esta herramienta solo trabaja en el directorio en el que se la ha llamado. Una vez dentro de la herramienta no se puede cambiar de directorio ni leer un fichero de otro directorio ni guardar ningún fichero en otro lado. Esta gestión de directorios cambiará en la segunda versión.

6.9- Planos

Desgraciadamente, al desarrollar esta versión no había ninguna herramienta disponible para poder diseñarla mediante el lenguaje ANA, ya que esta es la primera versión. Por lo tanto no se pudo hacer ningún plano de esta versión.

Capítulo 7

DISEÑO VERSIÓN 2

La razón principal para hacer una segunda versión de la herramienta ANA fue debido a los problemas a los que aparecieron al implantar la primera versión: se partía de una base que solo había previsto la representación de diagramas básicos. Esto hizo que la resolución de los problemas que planteaba el mantenimiento de los bloques IF fueran muy arduos de resolver. Por ello se decidió hacer una segunda versión partiendo de cero. Con ello se quiere decir que se cambió todo empezando por la base, que son las estructuras de datos.

7.1- Reestructuración

Aquí vamos a ver los cambios de base con respecto a la primera versión.

7.1.1- Estructuras de datos

Las estructuras se han cambiado completamente: para ello se hizo esta versión. Ahora se tiene solo tres niveles de estructuras. Recordese que en la primera versión la estructura 'corfil' era únicamente una

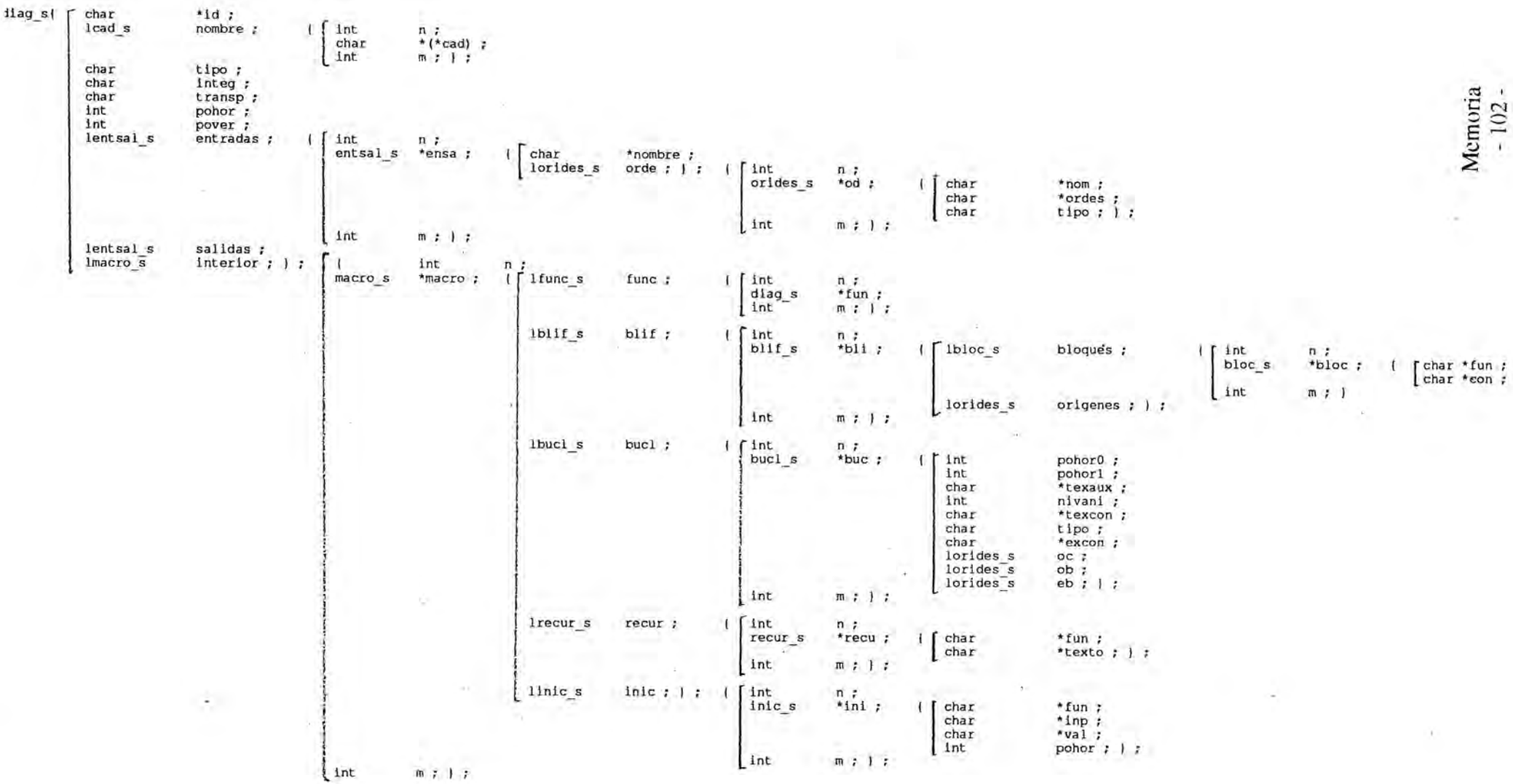
estructura de paso entre 'lectur y 'dibujo'. Estas dos estructuras ya tenían su razón de ser propias, como distintos niveles de definición del diagrama ANA.

Las nuevas estructuras son 'diagrama', 'dibujo' y 'grafic'. 'diagrama' es la de más alto nivel. Es la que habrá que mantener cuando se trabaje con ella y es tal como se almacenará en el fichero '.ANA'. Luego está 'dibujo' cuyo nivel es el mismo que en la versión anterior: el diagrama está representado en coordenadas de alto nivel que permite hacer cambios de escala sin tener que volver a calcularla. Y por fin la última es 'grafic': esta prácticamente no ha experimentado ningún cambio. A continuación describimos las peculiaridades de cada estructura.

La nueva estructura 'diagrama' se ha diseñado pensando en su mantenimiento posterior, con la experiencia adquirida en la realización de la primera versión. Por ello todas las funciones, que sean públicas o privadas (en la versión anterior serían cajas simples o compuestas) se han almacenado en una misma lista. En esta lista aparece incluso la función exterior: esto nos permite tratar mucho más racionalmente las entradas y las salidas de las funciones sin tener que particularizar cada vez para cada tipo de función. Luego se ha hecho que la estructura sea recursiva: este el cambio más importante incorporado a esta estructura. Cada función tiene un campo que es su interior en el que se describe otro dibujo independiente de su padre. Se puede de esta manera integrar muy fácilmente unos ficheros en otros y podemos representar en pantalla el

nivel de profundidad que se quiera. Ahora los bloques IF son funciones que se desarrollan interiormente y se integran y representan en pantalla solo si se desea. Pero son diagramas distintos el uno del otro y se mantienen por separado.

Otro cambio importante en el mantenimiento de la estructura es la gestión de memoria. Ahora cada puntero tiene una lista asignada que permite mantener con gran seguridad la memoria del puntero: se sabe en cada momento cuanta memoria está asignada a ese puntero y cuantos están asignados. A continuación se puede ver una representación de la estructura 'diagrama'.


diag_s
char *id ;
lcad_s nombre ; (int n ; char *(*cad) ; int m ;) ;
char tipo ;
char integ ;
char transp ;
int pohor ;
int pover ;
lentsal_s entradas ; (int n ; entsal_s *ensa ; (char *nombre ; lorides_s orde ;) ; (int n ; orides_s *od ; (char *nom ; char *ordes ; char tipo ;) ; int m ;) ; int m ;) ;
lentsal_s salidas ;
lmacro_s interior ;) ; (int n ; macro_s *macro ; (lfunc_s func ; (int n ; diag_s *fun ; int m ;) ;
lblif_s blif ; (int n ; blif_s *bli ; (lbloc_s bloques ; (int n ; bloc_s *bloc ; (char *fun ; char *con ; int m ;) ; lorides_s origenes ;) ; int m ;) ;
lbucl_s bucl ; (int n ; bucl_s *buc ; (int pohor0 ; int pohor1 ; char *texaux ; int nivani ; char *texcon ; char tipo ; char *excon ; lorides_s oc ; lorides_s ob ; lorides_s eb ;) ; int m ;) ;
lrecur_s recur ; (int n ; recur_s *recu ; (char *fun ; char *texto ;) ; int m ;) ;
linic_s inic ;) ; (int n ; inic_s *ini ; (char *fun ; char *inp ; char *val ; int pohor ;) ; int m ;) ;
int m ;) ;

La nueva estructura 'dibujo' se ha adaptado a las nuevas necesidades impuestas por la estructura 'diagrama'. Además se ha incluido en esta estructura los elementos de 'corfil' necesarios. De esta manera se ha eliminado una estructura transitoria inútil.

Para adaptarla a 'diagrama' se ha hecho una estructura recursiva. Recuerdese que a este nivel todavía se está en coordenadas de alto nivel, es decir coordenadas relativas. Por lo tanto se mantiene la independencia entre los distintos niveles de profundidad.

Por lo demás la única incorporación ha sido que cada elemento tiene asignado un color. Esto es para que se pueda representar los distintos elementos que conforman el diagrama en distintos colores: por ejemplo todas las funciones simples de un color, y las públicas de otro. Esto permite un más fácil comprensión del dibujo.

La gestión de memoria se ha tratado de la misma forma que en la estructura 'diagrama'. A continuación podemos ver la nueva estructura 'dibujo'.

dibujo_s
inter_s
inter;
liol_s
entradas;
int
liol_s
n;
*iol;
char
int
inpout_s
*fun;
orden;
inpout;
char
coohor_s
*nom;
ch;
coover_s
cv;
int sr;
int tw;
int hu;
int sr;
int th;
int vu;
liol_s
lfiedat_s
salidas;
lfiedat;
int
lfiedat_s
n;
*lfiedat;
liop_s
liop_s
ori;
des;
int
inpout_s
**inpout;
m;
linterf_s
linterf;
int
int
interf_s
m;
n;
*interf;
char
inter_s
*fun;
inter;
int
m;
marca_s
*marca;
coohor_s *ch[2];
coover_s *cv;
char num[3];
int color;
lgrupo_s
lgrupo;
int
grupo_s
n;
*grupo;
int
m;
pos_s
conjunto_s
pos;
conjunto;
coohor_s
coover_s
ch;
cv;

En cuanto a 'grafic' es básicamente la misma con la diferencia del añadido de los colores de cada elemento que ya esta incorporado en 'dibujo'. No es necesario por tanto representarla.

7.1.2- Cambios en la representación ANA

Existen varios cambios en la representación de los diagramas ANA en esta versión con respecto a la anterior.

Primero la nomenclatura cambia ligeramente: lo que antes se llamaban cajas compuestas y cajas simples ahora pasan a llamarse funciones públicas y privadas. Esto se ha hecho así para mantener una notación más rigurosa, acorde con el modelo función/variable. La función pública es aquella que puede ser llamada desde cualquier sitio al ser una función con entidad propia. La función privada es un grupo de instrucciones o de funciones agrupadas en una función que pertenece únicamente al diagrama en que se encuentra. Puede haber ditintos motivos para definir una función como privada. Por ejemplo, que su descripción interna sea irrelevante, por lo que se trata como una acción elemental.

Las funciones privadas pueden tener su interior descrito en el mismo diagrama. Este interior puede ser a su vez visible o no según

especifique el usuario. Es decir que la función puede ser una caja negra o ser transparente.

De esta manera se definen distintos niveles de profundidad en el dibujo que son representables según lo desee el usuario de la herramienta.

Al ser las funciones privadas algo perteneciente al diagrama y que no va a tener representación propia fuera de él, sus identificadores son todo lo largos que se quiera. Sin embargo las públicas tienen o pueden tener un fichero asociado lo que implica una limitación a ocho caracteres en el sistema operativo DOS en donde estamos trabajando. En cuanto a los identificadores, en la primera versión existía el problema de que no se podía poner el mismo identificador a dos funciones distintas por razones de mantenimiento internas. Ahora se puede hacer; el programa le pone al final del identificador una extensión que consta de un punto y de una letra para poder distinguir entre ellas. Estas extensiones solo serán visibles al entrar en modificar el diagrama (en ese momento es necesario poder distinguir entre ellas para especificar con cual se desea trabajar).

En la nueva versión se podrá tener un dato que salga de una función y que vaya al código. Esto no se podía hacer en la primera versión. La razón de incluir esta posibilidad es para mantener coherentemente el prototipo de una función. Existe la posibilidad de llamar a una función sin utilizar todos los datos que devuelve. Sin embargo esta función lo

devuelve. Este dato sencillamente se pierde. La manera de representarlo es una flecha que acaba en el "aire", al igual que una inicialización es una flecha que empieza en el "aire".

Uno de los problemas de la representación anterior es que unos elementos tan importantes como los bucles resaltaban muy poco. Para resolver este problema se ha decidido que la línea de retorno de los bucles se represente en pantalla con triple grosor. De esta manera se verán claramente definidos los bucles y sus límites.

El mayor cambio en la representación ANA reside en los bloques IF. En la anterior versión se representaba cada opción como un rectángulo que englobaba a una serie de cajas. Cada rectángulo se denominaba 'macro'. Y los datos iban directamente a las funciones pertenecientes al bloque IF. Esto ha cambiado radicalmente. Ahora cada opción del bloque IF es una función privada con su propio identificador. Por lo tanto tienen entidad propia: los datos que se envían al bloque IF no van directamente a las funciones interiores. Van primero a la función privada. Incluso si se desea se puede dejar el bloque IF como función privada sin desarrollar. Estas funciones obedecen a los mismos criterios de niveles de profundidad que cualquier otra función privada.

7.2- Nuevas posibilidades

Al plantearnos hacer una nueva versión de la herramienta ANA, quisimos aprovechar la experiencia adquirida para ampliarla lo más posible. Durante la utilización de la primera versión se ha echado en falta el no poder representar las relaciones estáticas de las funciones y de las variables. Estas son realmente necesarias en el diseño de un programa. El árbol de funciones para saber en todo momento en que sitio del programa se esta trabajando y el estado actual del proyecto global, y el árbol de variables para saber con qué estructuras se está trabajando.

Por ello se decidió incluirlo en el desarrollo de esta versión. El árbol de funciones es simplemente el típico árbol de subrutinas (o llamadas a funciones) que ya ha sido explicado anteriormente. En esta versión se podría manejarlos y combinarlos. Se podrá ver un árbol hasta el nivel de profundidad que se desee y se podrá ver que funciones son las que llaman al nodo raíz.

El árbol de variables es menos intuitivo que el de funciones. A continuación vamos a estudiar el caso de árbol de variables y su representación (esta representación no es la que se va a adaptar a la herramienta pero es similar).

Cada variable en programación tiene dos elementos que la caracterizan: su nombre, con el que nos referimos a ella, y su tipo de almacenamiento. En la figura -7.1- podemos ver representada la variable

'consumo' cuyo tipo de almacenamiento es DOUBLE (variable en doble precisión de coma flotante, a la que se reservan cuatro bytes en Lenguaje C).

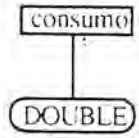


figura -7.1-

Representación de la variable 'consumo'

Otros tipo de almacenamiento simple es por ejemplo INT (un entero, dos bytes). Pero existen otros tipos de almacenamiento más complejos que vienen definidos por el programador: las estructuras de datos (esto está permitido en el Lenguaje C). En la figura -7.2- podemos ver la variable 'x' que pertenece al tipo COORDENADAS_S, que es una estructura que se compone de dos variables, 'ordenada' y 'abscisa' que son a su vez de tipo DOUBLE.

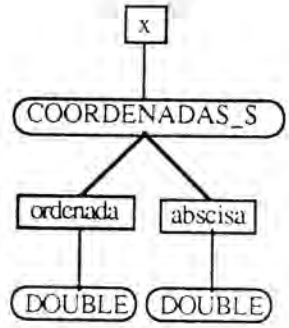


figura -7.2-

Representación de la estructura 'x'

Es posible que el tipo de las variables que constituyen una estructura sean a su vez estructuras, formando árboles. En la figura -7.3- podemos ver como la variable 'abscisa' es del tipo PARTE_ENTERA-DECIMAL_S, que se compone de dos campos de tipo INT, 'entera y 'decimal'.

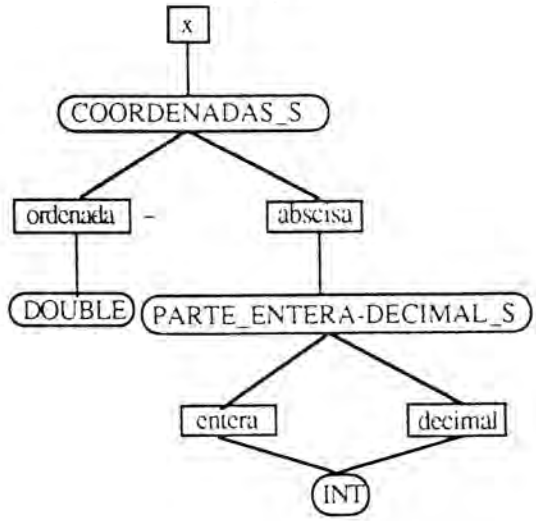


figura -7.3-

Representación del árbol de variables 'x'

De esta manera se pueden crear árboles de gran complejidad. En la figura -7.4- podemos ver el ejemplo de un árbol de variables real. Pertenece a un proyecto desarrollado por Fernando de Cuadra y Antonio Fernández en el I.I.T. (ver bibliografía: Cuadra, Fernández). Representa el estado del sistema de trenes de una línea de metro.

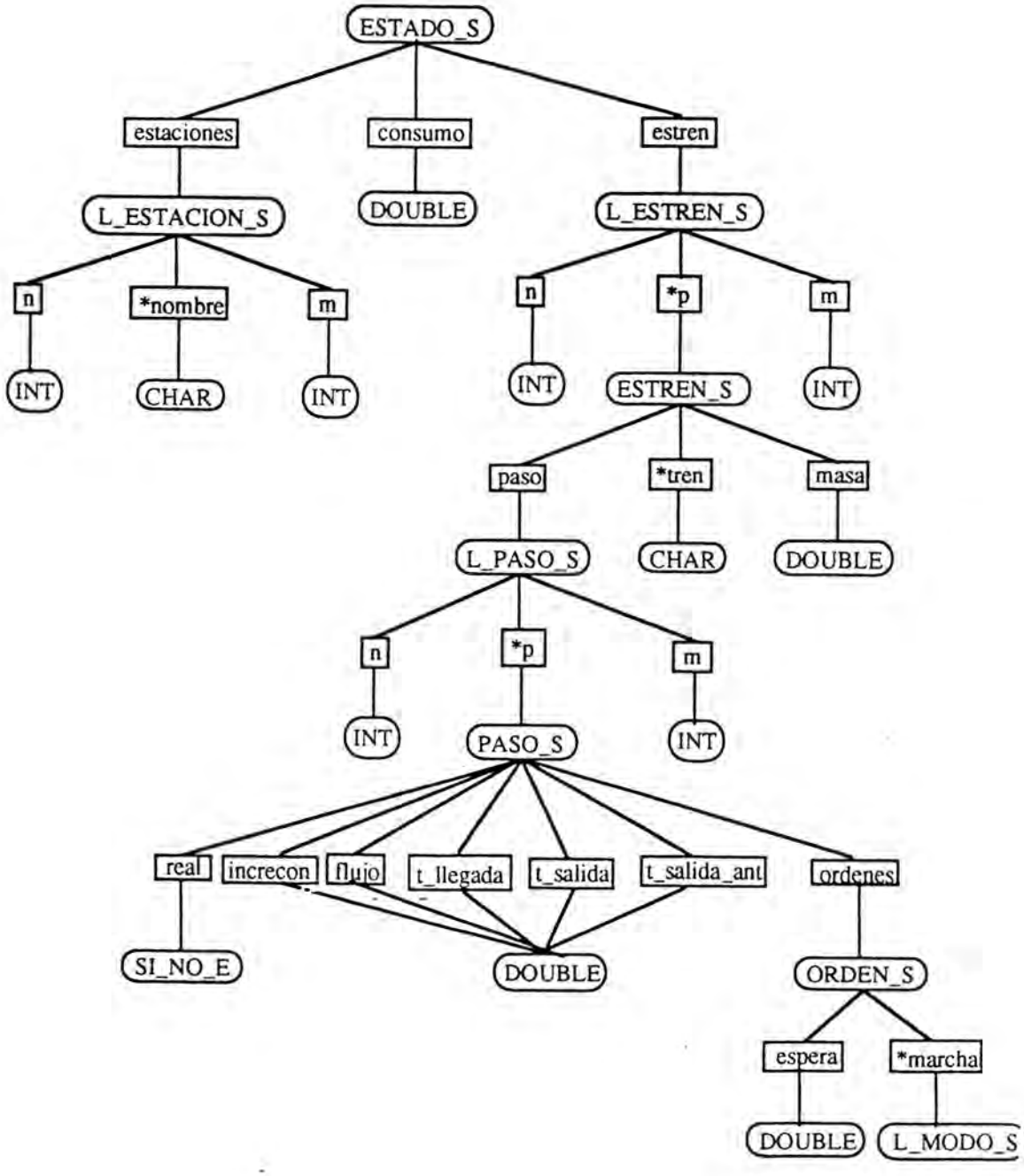


figura -7.4-

Representación del árbol del tipo de almacenamiento ESTADO_S

7.2.1- Entornos

En esta herramienta existen dos entornos bien diferenciados. Cada cual tiene su propio árbol de menús independiente del otro. Uno es el que todos conocemos: el mismo que la primera versión, en el cual se maneja la representación de las relaciones dinámicas de funciones y variables, creando lo que llamamos diagramas ANA. El otro se ha diseñado para la representación de los árboles de funciones y de variables o dicho de otro modo las relaciones estáticas. En este segundo entorno se incluirá la posibilidad de crear los prototipos de las funciones para luego poder verificar la consistencia de un árbol de funciones.

7.2.2- Estados

Cuando se utiliza la herramienta, se viaja constantemente por los árboles de menús, incluso cambiando de entorno. En cada sitio nos encontramos en estados distintos: podemos emprender un número de acciones limitadas y definidas para cada caso. Se pueden hacer dos grandes grupos de estados: los estados generales y los transitorios.

Un estado es transitorio cuando estamos realizando una acción que requiere seleccionar una o varias opciones (habría entonces varios

niveles de menús) pero que tiene una finalidad común. Todos estos estados por los que se pasa son considerados transitorios. Por ejemplo, si queremos poner un dato, debemos ir a la opción de poner datos. Una vez seleccionada, entramos en los estados transitorios: hay que elegir la función origen del dato, la función destino, el nombre de salida y el de llegada. Una vez acabado el proceso volvemos al estado inicial.

Los estados generales son los estados en no se ha emprendido ninguna acción particular que necesite de unas respuestas concretas. Así, el estado inicial del ejemplo anterior es un estado general ya que desde él se pueden emprender las acciones que se consideren pertinentes sin estar obligado a seguir una línea de preguntas. Esto es, por supuesto, relativo, ya que siempre estaremos limitados a las opciones de los distintos menús. Una peculiaridad de los estados generales es que estando en ellos se puede cambiar de entorno. La razón por la se limita a estos es para no dejar nunca una acción inacabada. También se accede al menú permanente sólo desde un estado general.

7.2.3- Estructura de estado

Esta versión tiene un seguimiento de la navegación por menús mucho más complicada que en la primera versión debido a la existencia de distintos entornos y el acceso a un menú permanente desde cualquier estado general. Por ello ha hecho falta hacer una estructura que nos diga

en dónde estamos, a dónde vamos y de dónde venimos. De esta manera podemos cambiar de estado y volver al mismo sitio en donde estábamos. El flujo del programa esta mucho más controlado y podemos quitar y poner los menús que hay en pantalla sin tener que pasar físicamente por su función: En la estructura tenemos almacenados los suficientes datos para rehacer la pantalla si ha habido que redibujar el diagrama de nuevo (por ejemplo al haber navegado por el dibujo).

7.3- Mantenimiento

La política de mantenimiento de la herramienta no varía con respecto a la versión anterior: solo se permitirá realizar cosas consistentes. En un diagrama nunca aparecerá algo imposible de programar.

Pero el mantenimiento del diagrama se ha simplificado enormemente con respecto a la versión anterior gracias a los cambios realizados. Se trabaja con un solo nivel a la vez, y ya no nos encontramos con problemas de anidamientos de bloques IF ni nada similar. Ahora modificar el interior de un bloque IF es sencillamente cambiar otro dibujo.

Otro elemento nuevo es la posibilidad de integrar varios niveles distintos, verificando la consistencia entre ellos, a modo de "linkador".

7.4- Menús

Los menús van a ser del mismo tipo que en la versión anterior. Internamente se harán más estándar para poder manejarlo con más soltura pero externamente serán los mismos.

Sin embargo aparecerá un nuevo menú. Este será de acceso permanente y tendrá las opciones de navegar por el dibujo, cambiarlo de escalar y cambiar de entorno. Este menú solo será accesible cuando nos encontremos en un estado denominado general.

7.5- Impresión

La impresión de los diagramas ANA será básicamente la misma. Además se podrán imprimir los árboles de funciones y de variables.

Un detalle importante es que se solucionarán los detalles del fichero PostScript que se cree para que éste sea del todo estándar y se pueda importar desde cualquier editor o programa de tratamiento de textos.

7.6- Configuración

Aquí habrá una notable mejora respecto a la versión anterior. Antes se podían cambiar todos los colores de la herramienta pero esos cambios

se perdían al abandonarla. Ahora estos se guardarán en un fichero personalizado de configuración, denominado 'ANACONFI.ACN'. Además se podrán configurar más cosas: la impresión por defecto de la herramienta, la longitud máxima de las cadenas de caracteres, etc. .

7.7- Política de ficheros

Las extensiones de los ficheros se seguirá conservando con respecto a la versión 1. Tendremos por lo tanto ficheros '.ANA', '.BK' y '.PS'. Lo que va a cambiar respecto de la otra versión va a ser los accesos a directorios. Antes solo se podía trabajar en el directorio en donde se estaba. Ahora se permitirá leer ficheros de otros directorios y guardar en otros directorios. Un fichero siempre se guardará en el directorio de donde se leyó a menos que el usuario especifique otra cosa.

Los ficheros de impresión (ficheros con extensión '.PS') se podrán dirigir a un directorio especificado en la configuración de la herramienta. Esto permitirá no mezclar todos los ficheros. Este directorio corresponde, por ejemplo al Output Directory del Turbo C.

Además existirá otro tipo de fichero con extensión '.PRJ' en el que se almacenarán todos los ficheros pertenecientes a un mismo proyecto, pudiendo estos pertenecer a distintos directorios. Esto será muy útil a la hora de verificar la consistencia del proyecto global.

7.8- Planos

En esta versión si disponemos de planos (diagramas ANA) del programa. Al tener la primera versión ya funcionando, se ha podido diseñar la herramienta con diagramas ANA y por tanto la tenemos especificada con esos diagramas. En el Documento nº2 (Planos) tenemos una selección de ellos.

Capítulo 8

GENERACIÓN DE CÓDIGO

En este capítulo se van a comentar ciertas características puntuales del código generado que nos parece pueden ser interesantes para el lector. Cada versión tiene sus propias peculiaridades, por supuesto.

8.1- Versión 1

- Las cuatro estructuras principales del programa ('lectur', 'corfil', 'dibujo' y 'grafic') son públicas en prácticamente todo el programa. Esto nos permite utilizarlas en cualquier momento sin preocuparnos de si lo hemos pasado como argumento o no. El problema que tiene esto es que utiliza mucha memoria ya que tiene que tenerlo siempre presente y además sólo puede haber una de cada con el mismo nombre (aparte de la falta de modularidad).

- El programa esta dividido en dos grandes árboles de funciones independientemente de las numerosas subrutinas: el árbol 'A' y el árbol 'ME'. El primero se ocupa de todo lo referente al dibujo y al fichero: leer el fichero, escribirlo, dibujarlo, guardarlo, etc.. .El segundo árbol se refiere a todo lo referente a los menús y lo que se desprende de ellos: aparición de menús, mantenimiento de lectur, etc.. .

- Como se ha podido comprobar en el párrafo anterior, se ha seguido el criterio ANA para organizar las funciones pero no se generó el código con un modelado en diagramas ANA: no había herramienta para ello.

- Gestión de memoria en esta versión: se hicieron funciones de reservar memoria para cada tipo de estructura que se tenía. Una para reservar memoria y otra para mantenerla. A esta función se le enviaba el número de elementos que se quería y un mensaje que es el aparecería en caso de error, para poder localizarlo. Para liberar se liberaba todo a la vez por completo, empezando por los niveles inferiores de las variables. Se tenía pocas funciones para ello.

8.2- Versión 2

- Un elemento importante es que esta versión se esta desarrollando en Standard C. De esta manera se espera que sea totalmente transportable para poder implantarla en otros entornos.

- Para poder realizar lo dicho anteriormente, se está aislando los módulos referentes al sistema operativo y a los gráficos ya que estos dependen totalmente del entorno en el que se esta trabajando. Esto es para poder reemplazarlos sin tocar el núcleo del programa y así hacer versiones para distintos ordenadores, pensando en un hardware más potente.

- Para mayor limpieza en el código y para que sea Standard C se ha elaborado una política de ficheros distinta de la primera versión: el código va almacenado en ficheros con extensión '.c'. A cada uno de estos ficheros le corresponde un fichero con extensión '.fun' en el que se almacenan los prototipos de todas las funciones definidas en el otro fichero. Aparte estos tenemos los ficheros con extensión '.var' en donde se almacenan todas las variables, concretamente las estructuras principales. Estos dos tipos de ficheros estarán incluidos como cabeceras en el código en los ficheros donde hagan falta.

- En esta versión se eliminan las variables públicas. Ahora todo se pasa por argumentos. Esto hace que se ocupe menos memoria y en cada función se sabe que se utiliza y que no se utiliza. Esto permite más claridad al comprender el programa y lo hace mucho más modular.

- La gestión de memoria es completamente distinta: como ya se ha explicado en el capítulo 7 ahora los punteros se auto mantienen con el método de las listas. Ahora en una función incluimos el reservar y mantener. Para liberar tenemos funciones que liberan desde cualquier nivel hacía abajo. Lo hacen llamando a funciones que liberan los niveles inferiores, todo perfectamente modular.

- En esta versión se utiliza la metodología ANA al completo, por lo tanto seguimos las recomendaciones de nombrar las funciones según el árbol al que pertenecen y las implementamos siguiendo los planos

realizados anteriormente con la herramienta a nuestra disposición: la primera versión de ANA. De esta manera se consigue un programa totalmente modular y sin ambigüedad en la nomenclatura de las funciones.

Capítulo 9

CONCLUSIÓN

9.1- Conclusiones

De este proyecto ha aportado muchos aspectos positivos: aprendizaje del lenguaje C que esta considerado como uno de los lenguajes del futuro por mucha gente y el diseño y la planificación de los programas de grandes dimensiones. Además se ha adquirido experiencia en una metodología de trabajo particular para abordar problemas complejos de programación.

Además es gratificante ver que el trabajo realizado da sus frutos: la primera versión de la herramienta ya esta funcionando y esta siendo utilizada para diseñar y especificar muchos proyectos de software en el I.I.T. (Instituto de Investigación Tecnológica). La acogida a este tipo de especificación no ha sido buena solo en el I.I.T. sino que todos los que han entrado en contacto con ella han estado muy interesados. Esto ha sido un gran aliento para desarrollar la segunda versión que se espera tenga todavía más éxito que la primera.

9.2- Desarrollos futuros

Al haber diseñado la segunda versión con módulos independientes y aislados, se podrá realizar fácilmente otras versiones en otros entornos: esto es alentador porque se ha visto en la primera versión que las limitaciones de memoria de un ordenador PC compatible son grandes, sobre todo para un programa tan grande como este.

En cuanto a las futuras aplicaciones de este programa, son muy extensas. Al tener los árboles de funciones incorporados, esto permitiría que la herramienta haga un análisis del programa en su conjunto. Se podría hacer que en una futura versión la herramienta generase pseudocódigo. Esto sería posible con un Estándar ANA diseñado para Standard C. Esto permitiría programar los grandes rasgos del programa directamente en un lenguaje gráfico de muy alto nivel y lo que se desearía desarrollar por sí mismo meterlo en funciones privadas que generarían una línea de comentarios en el pseudocódigo (por eso se llama así). Luego estas deberían ser desarrolladas para obtener el código definitivo. Pero se ahorraría el engorroso paso de diagramas ANA a código ya que el paso creativo ya se ha dado al diseñar los diagramas.

Bibliografía

Cuadra García, Fernando de:

"El problema general de la optimización de diseño por ordenador: aplicación de técnicas de Ingeniería del Conocimiento"
Tesis doctoral, Escuela Técnica Superior de Ingenieros Industriales, Universidad Pontificia Comillas, Madrid 1990.

Cuadra, Fernando de; Fernández, Antonio:

"Simulación y sistema de control de tráfico de un ferrocarril metropolitano. Herramienta de construcción automática de marchas óptimas para una interestación y tren dados."
Instituto de Investigación Tecnológica (I.I.T.), Universidad Pontificia Comillas, Madrid 1992.

Rodriguez-Piñeiro Fernández, Jorge:

"Cálculo infinitesimal"
I.C.A.I., Universidad Pontificia Comillas, Madrid 1986.

Systems Designers:

"CORE Manual (COntrolled Requirements Expression)"
Camberley (Surrey UK), 1985.

El importe de este proyecto asciende a la cantidad de nueve millones seiscientas cincuenta y nueve mil setecientas pesetas.

Madrid, a 2 de Junio de 1992

Firmado: Juan RIVIER ABBAD

CÁLCULOS

En este proyecto no se requieren cálculos

ESTUDIO ECONÓMICO

1- Estudio de mercado

Esta herramienta tiene un mercado casi ilimitado. La razón para ello es que para poder utilizarlo sólo se requiere un ordenador PC-compatible que esta al alcance de todo el mundo.

En cuanto a su utilización, hemos visto en la memoria que es útil en el desarrollo de prácticamente cualquier proyecto de Ingeniería del Conocimiento.

2- Ventas necesarias para amortizar el proyecto

A pesar de que esta herramienta tiene grandes posibilidades de comercialización debido al amplio mercado al que se destina, al desarrollarla no se tenía como finalidad el hacer un producto comercial. El principal objetivo del proyecto era tener una herramienta de ayuda para el diseño y especificación de algoritmos para uso interno del I.I.T. y que éste tuviese acceso al código d la herramienta: con se quiere decir que tuviese posibilidades de adaptarla a sus necesidades.

Por lo tanto el proyecto quedará totalmente amortizado cuando este acabado y se este utilizando en el I.I.T. (esto ya es verdad para la primera versión).

ANEJO I

SELECCIÓN DEL LISTADO DEL PROGRAMA VERSIÓN 1

```
#include "head.h"
#include "control.h"
#include <graphics.h>
#include <stdlib.h>

struct lectur       l       ;
struct corfil       c       ;
struct dibujo       d       ;
struct grafic       g       ;

/*-------------------------------------------------------------------*/
/*    A                    MAIN DE ANA                               */
/*-------------------------------------------------------------------*/

main( )

{

int                 gd = DETECT     ;
int                 gm      ;
double              fxy     ;
int                 pag     ;
int                 xymax[2];
int                 xyo[2]  ;
int                 facy[2] ;
int                 cte[2]  ;
int                 pm[2]   ;
int                 co[2]   ;
int                 xasp    ;
int                 yasp    ;
char                dum[13] ;
struct unidad       una     ;
struct unidad       unm     ;
struct unidad       un      ;
struct impres       imp     ;
struct flujo        flu     ;
struct color        col     ;
int                 me0( )  ;
int                 me1( )  ;
int                 me5( )  ;
int                 mes( )  ;
int                 meg( )  ;
int                 claved();
union inkey         {
                    char    ch[2]   ;
                    int     i       ;
                    }       ik      ;

if( registerbgidriver(EGAVGA_driver) < 0 ) exit( 1 ) ;
if( registerbgifont(triplex_font) < 0 ) exit( 1 ) ;
if( registerbgifont(small_font) < 0 ) exit( 1 ) ;

initgraph ( &gd, &gm , "" ) ;

col.dibu = WHITE ;
col.fond = CYAN ;
col.fove = LIGHTGRAY ;
col.reve = RED ;
col.teve = BLUE ;
```

```
col.tere = WHITE ;
col.inve = RED ;
col.reor = YELLOW ;

imp.fichero[0] = '\0' ;
strcpy( imp.font, "Times-Roman" ) ;          /*  INICIALICIACION  */
imp.puntos = 9 ;                             /*    DE LOS DATOS   */
imp.pchu = 50 ;                              /*      PARA LA      */
imp.pcvu = 30 ;                              /*     IMPRESION     */
strcpy( imp.escala, "Autom tico" ) ;

xymax[0] = getmaxx( ) ;
xymax[1] = getmaxy( ) ;

caratula( xymax ) ;

settextstyle( SMALL_FONT, HORIZ_DIR, 1 ) ;
setusercharsize( 1, 1, 1, 1 ) ;
getaspectratio( &xasp, &yasp ) ;

una.th = textheight( "a" ) ;
una.tw = textwidth( "a" ) ;
una.hu = ( una.tw ) / 3 ;
una.vu = una.hu * xasp / yasp ;

settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 ) ;

unm.th = textheight( "a" ) ;
unm.tw = textwidth( "a" ) ;

flu.accion = -1 ;
flu.volver = 0 ;
flu.status = 0 ;
flu.flag2 = 0 ;
cte[0] = 0 ;
cte[1] = 0 ;

if( ( flu.accion = me0( &unm, xymax, col, &imp, &una, dum ) ) == 3 ) {
        closegraph( ) ;
        exit( 0 ) ;
        }

setbkcolor( col.fond ) ;

do      {

        if( flu.status && flu.accion && ( flu.volver != -2 ) )
                do      {
                        flu.accion = -1 ;
                        while( !bioskey( 1 ) ) ;
                        ik.i = bioskey( 0 ) ;
                        if( ik.ch[0] == 0 )
                                flu.accion = claved( ik.ch[1], &pag, co ) ;
                        if( flu.accion == -1 )
                                flu.accion = me1( &flu.flag2, &unm, xymax, col ) ;
                                                                        /*printf("\n Despues
de me1 Memoria = %u ", coreleft( ) ) ;*/
                        }
                while( (flu.accion == -1) || (flu.accion == -4) ) ;
        else if( flu.status )
                flu.accion = 2 ;
```

```
        if( flu.accion < 5 ) {

                switch( flu.accion ) {
                        case 0 :
                                if( flu.status = = 1 ) {
                                        if( flu.flag2 = = 1 )
                                                if( mes( &unm, xymax, col ) = = 1 ) {
                                                        meg( &unm, xymax, col ) ;
                                                        flu.flag2 = 0 ;
                                                        }               /*printf("\n Antes de freeg
Memoria = %u , %d , %d ", coreleft( ), flu.accion , flu.status ) ;*/
                                                freeg( ) ;              /*printf("\n Antes de freed Memoria
= %u , %d , %d ", coreleft( ), flu.accion , flu.status ) ;*/
                                                freed( ) ;              /*printf("\n Antes de freel Memoria
= %u , %d , %d ", coreleft( ), flu.accion , flu.status ) ;*/
                                                freel( ) ;
                                                }                       /*printf("\n Despues de free Memoria
= %u , %d , %d ", coreleft( ), flu.accion , flu.status ) ;*/
                                        inil( ) ;                       /*getch( ) ;*/
                                        flu.opc = 0 ;                   /*exit( 1 ) ;*/
                                        cte[0] = 0 ;
                                        cte[1] = 0 ;
                                        flu.flag1 = 0 ;
                                        break ;
                        case 1 :                                        /*printf("\n Antes de me4 Memoria = %u
", coreleft( ) ) ;*/
                                        if( flu.status = = 1 ) {
                                                if( me4( &unm, xymax, col, dum ) = = -1 ) {
                                                        flu.opc = -1 ;
                                                        break ;
                                                        }               /*printf("\n Despues de me4
Memoria = %u ", coreleft( ) ) ;*/
                                                if( flu.flag2 = = 1 ) {
                                                        if( mes( &unm, xymax, col ) = = 1 )
                                                                meg( &unm, xymax, col ) ;
                                                        flu.flag2 = 0 ;
                                                        }               /*printf("\n Antes de freeg
Memoria = %u , %d , %d ", coreleft( ), flu.accion , flu.status ) ;*/
                                                freeg( ) ;              /*printf("\n Antes de freed Memoria
= %u , %d , %d ", coreleft( ), flu.accion , flu.status ) ;*/
                                                freed( ) ;              /*printf("\n Antes de freel Memoria
= %u , %d , %d ", coreleft( ), flu.accion , flu.status ) ;*/
                                                freel( ) ;              /*printf("\n Despues de free Memoria
= %u , %d , %d ", coreleft( ), flu.accion , flu.status ) ;*/
                                                }                       /*getch( ) ;printf("\nVoy a empezar a
leer") ;*/
                                        a0( dum ) ;                     /*printf("\n Despues de leer I Memoria
= %u , %d , %d ", coreleft( ), flu.accion , flu.status ) ;*/
                                        strcpy( imp.fichero, dum ) ;
                                        flu.opc = 0 ;
                                        cte[0] = 0 ;
                                        cte[1] = 0 ;
                                        flu.flag1 = 0 ;
                                        break ;
                        case 2 :
                                        a8( &g, facy, cte, col ) ;
                                        freeg( ) ;
                                        freed( ) ;
                                        flu.volver = me5( &unm, xymax, col ) ;
                                        strcpy( imp.fichero, I.id ) ;
```

```
                                        strcat( imp.fichero, ".ana" ) ;
                                        flu.opc = 0 ;
                                        flu.flag2 = 1 ;
                                        break ;
                                case 3 :                                /*printf("\n Antes de me5 Memoria = %u
", coreleft( ) ) ;*/
                                        if( a5( &d.dr, &una, &un, &unm, xymax, facy, col ) == -1 ) {
                                                flu.opc = -1 ;
                                                break ;
                                                }                               /*printf("\n Despues de me5 Memoria
= %u ", coreleft( ) ) ;printf("\n Antes de freeg Memoria = %u , %d , %d ", coreleft( ), flu.accion , flu.status
) ;*/
                                        freeg( ) ;                      /*printf("\n Despues de free Memoria =
%u , %d , %d ", coreleft( ), flu.accion , flu.status ) ;*/
                                        flu.flag1 = 1 ;
                                        flu.opc = 1 ;
                                        cte[0] = 0 ;
                                        cte[1] = 0 ;
                                        break ;
                                case 4 :                                /*printf("\n Antes de me3 Memoria = %u
", coreleft( ) ) ;*/
                                        if( me3( &unm, xymax, &col ) == -1 ) {
                                                flu.opc = -1 ;                  /*printf("\n Despues de me3
Memoria = %u ", coreleft( ) ) ;getch( ) ;*/
                                                break ;
                                                }                               /*printf("\n Despues de me3 Memoria
= %u ", coreleft( ) ) ;getch( ) ;*/
                                        setbkcolor( col.fond ) ;
                                        flu.opc = 2 ;
                                        break ;
                                case -2 :
                                        a6( xymax, xyo, cte, pm, col ) ;
                                        flu.opc = 2 ;
                                        break ;
                                case -3 :
                                        a7( xymax, xyo, cte, pm, pag ) ;
                                        flu.opc = 2 ;
                                }

                        switch( flu.opc ) {

                                case -1 :
                                        break ;
                                case 0 :
                                        a2( ) ;                         /*printf("\n despues de c y d Memoria =
%u , %d , %d ", coreleft( ), flu.accion , flu.status ) ;*/
                                case 1 :                                /*if(flu.status==1){closegraph( ) ;exit( 1 )
;}*/
                                        a3( &una, &un, facy, flu.flag1, pm, 0 ) ;   /*printf("\n despues de g
Memoria = %u , %d , %d ", coreleft( ), flu.accion , flu.status ) ;*/
                                case 2 :                                /*if(flu.status==1){closegraph( ) ;exit( 1 )
;}*/
                                        if( flu.accion == 2 )
                                                a9( pm, cte, xymax ) ;
                                        a4( &g, facy, cte, col ) ;
                                        flu.status = 1 ;
                                }
                        }
                else if( flu.accion == 5 )
                        impresion( &unm, xymax, col, &imp, &una, &un, facy, flu.flag1, pm ) ;
                else if( flu.accion == 6 ) {
```

```
                meg( &unm, xymax, col ) ;
                flu.flag2 = 0 ;
                }

        else    {
                if( flu.flag2 == 1 )
                        if( mes( &unm, xymax, col ) == 1 )
                                a1( ) ;
                break ;
                }

        if( pm[0] < (xymax[0]+3) )
                co[0] = 0 ;
        else
                co[0] = 1 ;

        if( pm[1] < (xymax[1]+3) )
                co[1] = 0 ;
        else
                co[1] = 1 ;

        }
while( 1 ) ;

closegraph( ) ;

}


/*-------------------------------------------------------------------*/
/*    A9                    VERIFICACION DEL TAMA¥O DEL DIBUJO       */
/*-------------------------------------------------------------------*/

a9( pm, cte, xymax )

int             pm[]    ;
int             cte[]   ;
int             xymax[] ;

{


if( pm[0] - cte[0] < xymax[0] )
        cte[0] = pm[0] - xymax[0] ;
cte[0] = ( cte[0] < 0 ) ? 0 : cte[0] ;

if( pm[1] - cte[1] < xymax[1] )
        cte[1] = pm[1] - xymax[1] ;
cte[1] = ( cte[1] < 0 ) ? 0 : cte[1] ;

}
```

```
#include "head.h"
#include <stdio.h>
#include <dir.h>
#include <dos.h>

extern struct lectur            l           ;

/*-------------------------------------------------------------------------*/
/*     A0          LECTURA DE FICHERO                                      */
/*-------------------------------------------------------------------------*/

a0( dum )

char                dum[13] ;

{

FILE                *fp         ;
int                 i           ;
struct cajita       *mem_cajita(int,char * );
struct blif         *mem_blif(int,char * );
struct bucli        *mem_bucli(int,char * );
struct cursi        *mem_cursi(int,char * );
struct inici        *mem_inici(int,char * );

                                                                    /*printf("\nNo he
hecho nada") ;*/
if( ( fp = fopen( dum, "r" ) ) == NULL )
        { closegraph( ) ; perror("error: a0: fopen: "); exit(1); }      /*printf("\nHe abierto el fichero") ;*/

a01( fp ) ;                                             /*printf("\nHe hecho a01") ;*/

a02( fp ) ;                                             /*printf("\nHe hecho a02") ;*/

if( l.ncajis > 0 ) {
        l.cs  = mem_cajita( l.ncajis, "a0(l.cs)" ) ;
        for( i=0 ; i < l.ncajis ; i++ )
                a03( fp, (l.cs+i), i ) ;
        }
                                                                    /*printf("\nHe hecho
a03") ;*/
if( l.ncajic > 0 ) {
        l.cc  = mem_cajita( l.ncajic, "a0(l.cc)" ) ;
        for( i=0 ; i < l.ncajic ; i++ )
                a04( fp, (l.cc+i), i ) ;
        }
                                                                    /*printf("\nHe hecho
a04") ;*/
if( l.nblif > 0 ) {
        l.bif = mem_blif( l.nblif, "a0(l.bif)" ) ;
        for( i=0 ; i < l.nblif ; i++ )
                a05( fp, (l.bif+i), i ) ;
        }
                                                                    /*printf("\nHe hecho
a05") ;*/
if( l.nbuc > 0 ) {
        l.buc = mem_bucli( l.nbuc, "a0(l.buc)" ) ;
        for( i=0 ; i < l.nbuc ; i++ )
                a06( fp, (l.buc+i), i ) ;
        }
```

```
                                                                    /*printf("\nHe hecho
a06") ;*/
if( l.nrecur > 0 ) {
        l.rec = mem_cursi( l.nrecur, "a0(l.rec)" ) ;
        for( i=0 ; i < l.nrecur ; i++ )
                a07( fp, (l.rec+i), i ) ;
        }
                                                                    /*printf("\nHe hecho
a07") ;*/
if( l.ninic > 0 ) {
        l.ini = mem_inici( l.ninic, "a0(l.ini)" ) ;
        for( i=0 ; i < l.ninic ; i++ )
                a08( fp, (l.ini+i), i ) ;
        }
                                                                    /*printf("\nHe hecho
a07") ;*/
fclose( fp ) ;
                                                                    /*printf("\nHe hecho
todo") ;*/
}

/*-------------------------------------------------------------------*/
/*    A01        LECTURA DE CAJA CABECERA                          */
/*-------------------------------------------------------------------*/

a01( fp )

FILE            *fp     ;

{

char            dum[40] ;
int             n       ;
int             i       ;
int             j       ;
int             k       ;
int             nn      ;
struct entrad   *sal    ;
struct salida   *ent    ;
int             lc0( )  ;
int             li0( )  ;
char            *mem_char(int,char * );
char            *(*mem_charp(int,char * ));
struct salida   *mem_salida(int,char * );
struct entrad   *mem_entrad(int,char * );


j = lc0( fp, dum ) ;

l.tree.id = mem_char( j+1, "a01(l.tree.id)" ) ;

ig0( l.tree.id, dum, j ) ;

l.tree.ncadna = li0( fp ) ;

n = l.tree.ncadna ;

if( n > 0 )
        l.tree.name = mem_charp( n, "a01(l.tree.name)" ) ;
```

```
for( i=0 ; i < n ; i++ ) {

        j = lc0( fp, dum ) ;

        *(l.tree.name+i) = mem_char( j+1, "a01(*l.tree.name)" ) ;

        ig0( *(l.tree.name+i), dum, j ) ;
        }

l.tree.pohor = li0( fp ) ;
l.tree.pover = li0( fp ) ;
l.tree.nentra = li0( fp ) ;
l.tree.nsalid = li0( fp ) ;

n = l.tree.nentra ;

if( n > 0 )
        l.tree.ent = mem_salida( n, "a01(l.tree.ent)" ) ;

for( i=0 ; i < n ; i++ ) {

        ent = (l.tree.ent+i) ;

    j = lc0( fp, dum ) ;

        (l.tree.ent+i)->nam = mem_char( j+1, "a01(l.tree.ent->name)" ) ;

        ig0( ent->nam, dum, j ) ;

        ent->norig = li0( fp ) ;

        nn = ent->norig ;
        (l.tree.ent+i)->nom  = mem_charp( nn, "a01(l.tree.ent->nom)" ) ;
        (l.tree.ent+i)->ori  = mem_charp( nn, "a01(l.tree.ent->ori)" ) ;
        (l.tree.ent+i)->tipo = mem_char( nn, "a01(l.tree.ent->tipo)" ) ;

        for( k=0 ; k < nn ; k++ ) {

                j = lc0( fp, dum ) ;

                *((l.tree.ent+i)->nom+k) = mem_char( j+1, "a01(*l.tree.ent->nom)" ) ;

                ig0( *(ent->nom+k), dum, j ) ;

                j = lc0( fp, dum ) ;

                *((l.tree.ent+i)->ori+k) = mem_char( j-1, "a01(*l.tree.ent->ori)" ) ;

                *(ent->tipo+k) = dum[0] ;

                ig0( *(ent->ori+k), dum+2, j-2 ) ;
                }
        }

n = l.tree.nsalid ;

if( n > 0 )
        l.tree.sal = mem_entrad( n, "a01(l.tree.sal)" ) ;

for( i=0 ; i < n ; i++ ) {
```

```
        sal = (l.tree.sal+i) ;

    j = lc0( fp, dum ) ;

        (l.tree.sal+i)->nam = mem_char( j+1, "a01(l.tree.sal->nam)" ) ;

        ig0( sal->nam, dum, j ) ;

        sal->ndest = li0( fp ) ;

        nn = sal->ndest ;
        (l.tree.sal+i)->nom  = mem_charp( nn, "a01(l.tree.sal->nom)" ) ;
        (l.tree.sal+i)->des  = mem_charp( nn, "a01(l.tree.sal->des)" ) ;
        (l.tree.sal+i)->tipo = mem_char( nn, "a01(l.tree.sal->tipo)" ) ;

        for( k=0 ; k < nn ; k++ ) {

                j = lc0( fp, dum ) ;

                *((l.tree.sal+i)->nom+k) = mem_char( j+1, "a01(*l.tree.sal->nom)" ) ;

                ig0( *(sal->nom+k), dum, j ) ;

                j = lc0( fp, dum ) ;

                *((l.tree.sal+i)->des+k) = mem_char( j-1, "a01(*l.tree.sal->des)" ) ;

                *(sal->tipo+k) = dum[0] ;

                ig0( *(sal->des+k), dum+2, j-2 ) ;
                }
        }
}

/*-------------------------------------------------------------------*/
/*    A02          LECTURA DE RELACIONES INTERNAS                    */
/*-------------------------------------------------------------------*/

a02( fp )

FILE            *fp     ;

{

char            dum[40] ;
int             n       ;
int             i       ;
int             j       ;
int             k       ;
int             nn      ;
struct entrad   *ent    ;
struct salida   *sal    ;
int             lc0( )  ;
int             li0( )  ;
char            *mem_char(int,char * );
char            *(*mem_charp(int,char * ));
struct salida   *mem_salida(int,char * );
struct entrad   *mem_entrad(int,char * );
```

```
j = lc0( fp, dum ) ;

l.id = mem_char( j+1, "a02(l.id)" ) ;

ig0( l.id, dum, j ) ;

j = lc0( fp, dum ) ;

l.name = mem_char( j+1, "a02(l.name" ) ;

ig0( l.name, dum, j ) ;

l.nentra = li0( fp ) ;
l.nsalid = li0( fp ) ;
l.ncajis = li0( fp ) ;
l.ncajic = li0( fp ) ;
l.nblif  = li0( fp ) ;
l.nbuc   = li0( fp ) ;
l.nrecur = li0( fp ) ;
l.ninic  = li0( fp ) ;

n = l.nentra ;

if( n > 0 )
        l.ent = mem_entrad( n, "a02(l.ent)" ) ;

for( i=0 ; i < n ; i++ ) {

        ent = (l.ent+i) ;

    j = lc0( fp, dum ) ;

        (l.ent+i)->nam = mem_char( j+1, "a02(l.ent->nam)" ) ;

        ig0( ent->nam, dum, j ) ;

        ent->ndest = li0( fp ) ;

        nn = ent->ndest ;
        (l.ent+i)->nom  = mem_charp( nn, "a02(l.ent->nom)" ) ;
        (l.ent+i)->des  = mem_charp( nn, "a02(l.ent->des)" ) ;
        (l.ent+i)->tipo = mem_char( nn, "a02(l.ent->tipo)" ) ;

        for( k=0 ; k < nn ; k++ ) {

                j = lc0( fp, dum ) ;

                *((l.ent+i)->nom+k) = mem_char( j+1, "a02(*l.ent->nom)" ) ;

                ig0( *(ent->nom+k), dum, j ) ;

                j = lc0( fp, dum ) ;

                *((l.ent+i)->des+k) = mem_char( j-1, "a02(*l.ent->des)" ) ;

                *(ent->tipo+k) = dum[0] ;

                ig0( *(ent->des+k), dum+2, j-2 ) ;
                }
        }
```

```
n = l.nsalid ;

if( n > 0 )
        l.sal = mem_salida( n, "a02(l.sal)" ) ;

for( i=0 ; i < n ; i++ ) {

        sal = (l.sal+i) ;

    j = lc0( fp, dum ) ;

        (l.sal+i)->nam = mem_char( j+1, "a02(l.sal->nam)" ) ;

        ig0( sal->nam, dum, j ) ;

        sal->norig = li0( fp ) ;

        nn = sal->norig ;
        (l.sal+i)->nom  = mem_charp( nn, "a02(l.sal->nom" ) ;
        (l.sal+i)->ori  = mem_charp( nn, "a02(l.sal->ori" ) ;
        (l.sal+i)->tipo = mem_char( nn, "a02(l.sal->tipo" ) ;

        for( k=0 ; k < nn ; k++ ) {

                j = lc0( fp, dum ) ;

                *((l.sal+i)->nom+k) = mem_char( j+1, "a02(*l.sal->nom)" ) ;

                ig0( *(sal->nom+k), dum, j ) ;

                j = lc0( fp, dum ) ;

                *((l.sal+i)->ori+k) = mem_char( j-1, "a02(*l.sal->ori)" ) ;

                *(sal->tipo+k) = dum[0] ;

                ig0( *(sal->ori+k), dum+2, j-2 ) ;
                }
        }
}

/*-------------------------------------------------------------------*/
/*    A03          LECTURA DE CAJA SIMPLE                            */
/*-------------------------------------------------------------------*/

a03( fp, cs, ii )

FILE            *fp     ;
struct cajita   *cs     ;
int             ii      ;

{

char            dum[40] ;
int             n       ;
int             i       ;
int             j       ;
int             k       ;
int             nn      ;
struct entrad   *sal    ;
```

```
struct salida       *ent     ;
int                 lc0( )   ;
int                 li0( )   ;
char                *mem_char(int,char * );
char                *(*mem_charp(int,char * ));
struct salida       *mem_salida(int,char * );
struct entrad       *mem_entrad(int,char * );


j = lc0( fp, dum ) ;

(l.cs+ii)->id = mem_char( j+1, "a03(l.cs->id)" ) ;

ig0( cs->id, dum, j ) ;

cs->ncadna = li0( fp ) ;

n = cs->ncadna ;

if( n > 0 )
        (l.cs+ii)->name = mem_charp( n, "a03(l.cs->name)" ) ;

for( i=0 ; i < n ; i++ ) {

        j = lc0( fp, dum ) ;

        *((l.cs+ii)->name+i) = mem_char( j+1, "a03(*l.cs->name)" ) ;

        ig0( *(cs->name+i), dum, j ) ;
        }

cs->pohor = li0( fp ) ;
cs->pover = li0( fp ) ;
cs->nentra = li0( fp ) ;
cs->nsalid = li0( fp ) ;

n = cs->nentra ;

if( n > 0 )
        (l.cs+ii)->ent = mem_salida( n, "a03(l.cs->ent)" ) ;

for( i=0 ; i < n ; i++ ) {

        ent = (cs->ent+i) ;

    j = lc0( fp, dum ) ;

        ((l.cs+ii)->ent+i)->nam = mem_char( j+1, "a03(l.cs->ent->nam)" ) ;

        ig0( ent->nam, dum, j ) ;

        ent->norig = li0( fp ) ;

        nn = ent->norig ;
        ((l.cs+ii)->ent+i)->nom  = mem_charp( nn, "a03(l.cs->ent->nom)" ) ;
        ((l.cs+ii)->ent+i)->ori  = mem_charp( nn, "a03(l.cs->ent->ori)" ) ;
        ((l.cs+ii)->ent+i)->tipo = mem_char( nn, "a03(l.cs->ent->tipo)" ) ;

        for( k=0 ; k < nn ; k++ ) {

                j = lc0( fp, dum ) ;
```

```
                *(((l.cs+ii)->ent+i)->nom+k) = mem_char( j+1, "a03(*l.cs->ent->nom)" ) ;

                igO( *(ent->nom+k), dum, j ) ;

                j = lcO( fp, dum ) ;

                *(((l.cs+ii)->ent+i)->ori+k) = mem_char( j-1, "a03(*l.cs->ent->ori)" ) ;

                *(ent->tipo+k) = dum[0] ;

                igO( *(ent->ori+k), dum+2, j-2 ) ;
                }
        }

n = cs->nsalid ;

if( n > 0 )
        (l.cs+ii)->sal = mem_entrad( n, "a03(l.cs->sal)" ) ;

for( i=0 ; i < n ; i++ ) {

        sal = (cs->sal+i) ;

    j = lcO( fp, dum ) ;

        ((l.cs+ii)->sal+i)->nam = mem_char( j+1,"a03(l.cs->sal->nam)" ) ;

        igO( sal->nam, dum, j ) ;

        sal->ndest = liO( fp ) ;

        nn = sal->ndest ;
        ((l.cs+ii)->sal+i)->nom  = mem_charp( nn, "a03(l.cs->sal->nom)" ) ;
        ((l.cs+ii)->sal+i)->des  = mem_charp( nn, "a03(l.cs->sal->des)" ) ;
        ((l.cs+ii)->sal+i)->tipo = mem_char( nn, "a03(l.cs->sal->tipo)" ) ;

        for( k=0 ; k < nn ; k++ ) {

                j = lcO( fp, dum ) ;

                *(((l.cs+ii)->sal+i)->nom+k) = mem_char( j+1, "a03(*l.cs->sal->nom)" ) ;

                igO( *(sal->nom+k), dum, j ) ;

                j = lcO( fp, dum ) ;

                *(((l.cs+ii)->sal+i)->des+k) = mem_char( j-1, "a03(*l.cs->sal->des)" ) ;

                *(sal->tipo+k) = dum[0] ;

                igO( *(sal->des+k), dum+2, j-2 ) ;
                }
        }
}

/*-------------------------------------------------------------------*/
/*    A04          LECTURA DE CAJA COMPUESTA                         */
/*-------------------------------------------------------------------*/
```

```
a04( fp, cc, ii )

FILE            *fp     ;
struct cajita   *cc     ;
int             ii      ;

{

char            dum[40] ;
int             n       ;
int             i       ;
int             j       ;
int             k       ;
int             nn      ;
struct entrad   *sal    ;
struct salida   *ent    ;
int             lc0( )  ;
int             li0( )  ;
char            *mem_char(int,char * );
char            *(*mem_charp(int,char * ));
struct salida   *mem_salida(int,char * );
struct entrad   *mem_entrad(int,char * );


j = lc0( fp, dum ) ;

(l.cc+ii)->id = mem_char( j+1, "a04(l.cc->id)" ) ;

ig0( cc->id, dum, j ) ;

cc->ncadna = li0( fp ) ;

n = cc->ncadna ;

if( n > 0 )
        (l.cc+ii)->name = mem_charp( n, "a04(l.cc->name" ) ;

for( i=0 ; i < n ; i++ ) {

        j = lc0( fp, dum ) ;

        *((l.cc+ii)->name+i) = mem_char( j+1, "a04(*l.cc->name" ) ;

        ig0( *(cc->name+i), dum, j ) ;
        }

cc->pohor  = li0( fp ) ;
cc->pover  = li0( fp ) ;
cc->nentra = li0( fp ) ;
cc->nsalid = li0( fp ) ;

n = cc->nentra ;

if( n > 0 )
        (l.cc+ii)->ent = mem_salida( n, "a04(l.cc->ent)" ) ;

for( i=0 ; i < n ; i++ ) {

        ent = (cc->ent+i) ;

    j = lc0( fp, dum ) ;
```

```
        ((l.cc+ii)->ent+i)->nam = mem_char( j+1, "a04(l.cc->ent->nam)" ) ;

        igO( ent->nam, dum, j ) ;

        ent->norig = liO( fp ) ;

        nn = ent->norig ;
        ((l.cc+ii)->ent+i)->nom  = mem_charp( nn, "a04(l.cc->ent->nom)" ) ;
        ((l.cc+ii)->ent+i)->ori  = mem_charp( nn, "a04(l.cc->ent->ori)" ) ;
        ((l.cc+ii)->ent+i)->tipo = mem_char( nn, "a04(l.cc->ent->tipo)" ) ;

        for( k=0 ; k < nn ; k++ ) {

                j = lcO( fp, dum ) ;

                *(((l.cc+ii)->ent+i)->nom+k) = mem_char( j+1, "a04(*l.cc->ent->nom)" ) ;

                igO( *(ent->nom+k), dum, j ) ;

                j = lcO( fp, dum ) ;

                *(((l.cc+ii)->ent+i)->ori+k) = mem_char( j-1, "a04(*l.cc->ent->ori)" ) ;

                *(ent->tipo+k) = dum[0] ;

                igO( *(ent->ori+k), dum+2, j-2 ) ;
                }
        }

n = cc->nsalid ;

if( n > 0 )
        (l.cc+ii)->sal = mem_entrad( n, "a04(l.cc->sal)" ) ;

for( i=0 ; i < n ; i++ ) {

        sal = (cc->sal+i) ;

    j = lcO( fp, dum ) ;

        ((l.cc+ii)->sal+i)->nam = mem_char( j+1, "a04(l.cc->sal->nam)" ) ;

        igO( sal->nam, dum, j ) ;

        sal->ndest = liO( fp ) ;

        nn = sal->ndest ;
        ((l.cc+ii)->sal+i)->nom  = mem_charp( nn, "a04(l.cc->sal->nom)" ) ;
        ((l.cc+ii)->sal+i)->des  = mem_charp( nn, "a04(l.cc->sal->des)" ) ;
        ((l.cc+ii)->sal+i)->tipo = mem_char( nn, "a04(l.cc->sal->tipo)" ) ;

        for( k=0 ; k < nn ; k++ ) {

                j = lcO( fp, dum ) ;

                *(((l.cc+ii)->sal+i)->nom+k) = mem_char( j+1, "a04(*l.cc->sal->nom)" ) ;

                igO( *(sal->nom+k), dum, j ) ;

                j = lcO( fp, dum ) ;
```

```
                *(((l.cc+ii)->sal+i)->des+k) = mem_char( j-1, "a04(*l.cc->sal->des)" ) ;

                *(sal->tipo+k) = dum[0] ;

                igO( *(sal->des+k), dum+2, j-2 ) ;
                }
        }
}

/*-------------------------------------------------------------------------*/
/*    A05          LECTURA DE BLOQUES IF                                   */
/*-------------------------------------------------------------------------*/

a05( fp, bi, ii )

FILE            *fp     ;
struct blif     *bi     ;
int             ii      ;

{

char            dum[40] ;
int             n       ;
int             i       ;
int             j       ;
int             k       ;
struct macro    *ma     ;
int             lcO( )  ;
int             liO( )  ;
char            *mem_char(int,char * );
char            *(*mem_charp(int,char * ));
struct macro    *mem_macro(int,char * );

bi->hueh1   = liO( fp ) ;
bi->bifpa   = liO( fp ) ;
bi->macma   = liO( fp ) ;
bi->nivani  = liO( fp ) ;
bi->nmac    = liO( fp ) ;
bi->oc.norig = liO( fp ) ;

(l.bif+ii)->mac = mem_macro( bi->nmac, "a05(l.bif->mac)" ) ;

for( i=0 ; i < bi->nmac ; i++ ) {

        ma = (bi->mac+i) ;

        j = lcO( fp, dum ) ;

        ((l.bif+ii)->mac+i)->con = mem_char( j+1, "a05(l.bif->mac->con)" ) ;

        igO( ma->con, dum, j ) ;

        ma->nivcon = liO( fp ) ;
        ma->hueh1 = liO( fp ) ;
        ma->ncaj = liO( fp ) ;

        ((l.bif+ii)->mac+i)->caj = mem_charp( ma->ncaj, "a05(l.bif->mac->caj)" ) ;

        for( k=0 ; k < ma->ncaj ; k++ ) {
```

```
                j = lc0( fp, dum ) ;

                *((l.bif+ii)->mac+i)->caj+k) = mem_char( j+1, "a05(*l.bif->mac->caj)" ) ;

                ig0( *(ma->caj+k), dum, j ) ;
                }
        }

n = bi->oc.norig ;

if( n > 0 ) {
        (l.bif+ii)->oc.nom  = mem_charp( n, "a05(l.bif->oc.nom)" ) ;
        (l.bif+ii)->oc.ori  = mem_charp( n, "a05(l.bif->oc.ori)" ) ;
        (l.bif+ii)->oc.tipo = mem_char( n, "a05(l.bif->oc.tipo)" ) ;
        }

for( i=0 ; i < n ; i++ ) {

        j = lc0( fp, dum ) ;

        *((l.bif+ii)->oc.nom+i) = mem_char( j+1, "a05(*l.bif->oc.nom)" ) ;

        ig0( *(bi->oc.nom+i), dum, j ) ;

        j = lc0( fp, dum ) ;

        *((l.bif+ii)->oc.ori+i) = mem_char( j-1, "a05(*l.bif->oc.ori)" ) ;

        *(bi->oc.tipo+i) = dum[0] ;

        ig0( *(bi->oc.ori+i), dum+2, j-2 ) ;
        }
}

/*-------------------------------------------------------------------------*/
/*    A06          LECTURA DE BUCLES                                       */
/*-------------------------------------------------------------------------*/

a06( fp, bu, ii )

FILE            *fp     ;
struct bucli    *bu     ;
int             ii      ;

{

char            dum[40] ;
int             n       ;
int             i       ;
int             j       ;
int             lc0( )  ;
int             li0( )  ;
char            *mem_char(int,char * );
char            *(*mem_charp(int,char * ));


bu->ncaj = li0( fp ) ;

(l.buc+ii)->caj = mem_charp( bu->ncaj, "a05(l.buc->caj)" ) ;
```

```
for( i=0 ; i < bu->ncaj ; i++ ) {

        j = lc0( fp, dum ) ;

        *((l.buc+ii)->caj+i) = mem_char( j+1, "a05(*l.buc->caj)" ) ;

        ig0( *(bu->caj+i), dum, j ) ;
        }

j = lc0( fp, dum ) ;

(l.buc+ii)->texaux = mem_char( j+1, "a06(l.buc->texaux)" ) ;

ig0( bu->texaux, dum, j ) ;

j = lc0( fp, dum ) ;

(l.buc+ii)->texcon = mem_char( j+1, "a06(l.buc->texcon)" ) ;
ig0( bu->texcon, dum, j ) ;

bu->oc.norig = li0( fp ) ;
bu->ob.norig = li0( fp ) ;
bu->eb.norig = li0( fp ) ;
bu->nivcon   = li0( fp ) ;
bu->nivani   = li0( fp ) ;

j = lc0( fp, dum ) ;

(l.buc+ii)->excon = mem_char( j-1, "a06(l.buc->excon)" ) ;

bu->tipo = dum[0] ;

ig0( bu->excon, dum+2, j-2 ) ;

n = bu->oc.norig ;

if( n > 0 ) {
        (l.buc+ii)->oc.nom  = mem_charp( n, "a06(l.buc->oc.nom)" ) ;
        (l.buc+ii)->oc.ori  = mem_charp( n, "a06(l.buc->oc.ori)" ) ;
        (l.buc+ii)->oc.tipo = mem_char( n, "a06(l.buc->oc.tipo)" ) ;
        }

for( i=0 ; i < n ; i++ ) {

        j = lc0( fp, dum ) ;

        *((l.buc+ii)->oc.nom+i) = mem_char( j+1, "a06(*l.buc->oc.nom)" ) ;

        ig0( *(bu->oc.nom+i), dum, j ) ;

        j = lc0( fp, dum ) ;

        *((l.buc+ii)->oc.ori+i) = mem_char( j-1, "a06(*l.buc->oc.ori)" ) ;

        *(bu->oc.tipo+i) = dum[0] ;

        ig0( *(bu->oc.ori+i), dum+2, j-2 ) ;
        }

n = bu->ob.norig ;
```

```
if( n > 0 ) {
        (l.buc+ii)->ob.nom  = mem_charp( n, "a06(l.buc->ob.nom)" ) ;
        (l.buc+ii)->ob.ori  = mem_charp( n, "a06(l.buc->ob.ori)" ) ;
        (l.buc+ii)->ob.tipo = mem_char( n, "a06(l.buc->ob.tipo)" ) ;
        }

for( i=0 ; i < n ; i++ ) {

        j = lc0( fp, dum ) ;

        *((l.buc+ii)->ob.nom+i) = mem_char( j+1, "a06(*l.buc->ob.nom)" ) ;

        ig0( *(bu->ob.nom+i), dum, j ) ;

        j = lc0( fp, dum ) ;

        *((l.buc+ii)->ob.ori+i) = mem_char( j-1, "a06(*l.buc->ob.ori)" ) ;

        *(bu->ob.tipo+i) = dum[0] ;

        ig0( *(bu->ob.ori+i), dum+2, j-2 ) ;
        }

n = bu->eb.norig ;

if( n > 0 ) {
        (l.buc+ii)->eb.nom  = mem_charp( n, "a06(l.buc->eb.nom)" ) ;
        (l.buc+ii)->eb.ori  = mem_charp( n, "a06(l.buc->eb.ori)" ) ;
        (l.buc+ii)->eb.tipo = mem_char( n, "a06(l.buc->eb.tipo)" ) ;
        }

for( i=0 ; i < n ; i++ ) {

        j = lc0( fp, dum ) ;

        *((l.buc+ii)->eb.nom+i) = mem_char( j+1, "a06(*l.buc->eb.nom)" ) ;

        ig0( *(bu->eb.nom+i), dum, j ) ;

        j = lc0( fp, dum ) ;

        *((l.buc+ii)->eb.ori+i) = mem_char( j-1, "a06(*l.buc->eb.ori)" ) ;

        *(bu->eb.tipo+i) = dum[0] ;

        ig0( *(bu->eb.ori+i), dum+2, j-2 ) ;
        }

}

/*-------------------------------------------------------------------*/
/*    A07        LECTURA DE RECURSIONES                              */
/*-------------------------------------------------------------------*/

a07( fp, re, ii )

FILE            *fp     ;
struct cursi    *re     ;
int             ii      ;
```

```
{

char            dum[40] ;
int             j       ;
int             lc0( )  ;
char            *mem_char(int,char * );


j = lc0( fp, dum ) ;

(l.rec+ii)->caja = mem_char( j+1, "a07(l.rec->caj)" ) ;

ig0( re->caja, dum, j ) ;

j = lc0( fp, dum ) ;

(l.rec+ii)->tex = mem_char( j+1, "a07(l.rec->tex)" ) ;

ig0( re->tex, dum, j ) ;

}

/*-------------------------------------------------------------------*/
/*    A08          LECTURA DE INICIALIZACIONES                       */
/*-------------------------------------------------------------------*/

a08( fp, in, ii )

FILE            *fp     ;
struct inici    *in     ;
int             ii      ;

{

char            dum[40] ;
int             j       ;
int             lc0( )  ;
int             li0( )  ;
char            *mem_char(int,char * );


j = lc0( fp, dum ) ;

(l.ini+ii)->caja = mem_char( j+1, "a08(l.ini->caja)" ) ;

ig0( in->caja, dum, j ) ;

j = lc0( fp, dum ) ;

(l.ini+ii)->inp = mem_char( j+1, "a08(l.ini->inp)" ) ;

ig0( in->inp, dum, j ) ;

j = lc0( fp, dum ) ;

(l.ini+ii)->val = mem_char( j+1, "a08(l.ini->val)" ) ;

ig0( in->val, dum, j ) ;
```

```
j = lc0( fp, dum ) ;

(l.ini+ii)->anca = mem_char( j+1, "a08(l.ini->anca)" ) ;

ig0( in->anca, dum, j ) ;

}

/*-------------------------------------------------------------------*/
/*     A1          ESCRITURA EN FICHERO                              */
/*-------------------------------------------------------------------*/

a1( )

{

FILE            *fp     ;
char            dum[13] ;
char            dub[13] ;
int             i       ;
int             j       ;
struct ffblk    dir     ;
int     at = FA_ARCH    ;


strcpy( dum, l.id ) ;
strcat( dum, ".ana" ) ;

strcpy( dub, l.id ) ;
strcat( dub, ".bk" ) ;

if( !findfirst( dum, &dir, at ) ) {
        if( !findfirst( dub, &dir, at ) )
                if( remove( dub ) == -1 )
                        { closegraph( ) ; perror("error: a1: remove(*.bk): ") ; exit( 1 ) ; }
        if( ( rename( dum, dub ) ) == -1 )
                { closegraph( ) ; perror("error: a1: rename: ") ; exit( 1 ) ; }
        }

if( ( fp = fopen( dum, "w" ) ) == NULL )
        { closegraph( ) ; perror("error: a1: fopen: ") ; exit( 1 ) ; }

for( i=0 ; i<80 ; i++ )
        fprintf( fp, "=" ) ;
fprintf( fp, "\n\tFichero ANA :\t%s.ANA\t< como caja compuesta >\n",l.id) ;
for( i=0 ; i<80 ; i++ )
        fprintf( fp, "=" ) ;
fprintf( fp, "\n\n") ;

a11( fp ) ;

fprintf( fp, "\n") ;
for( i=0 ; i<80 ; i++ )
        fprintf( fp, "=" ) ;
fprintf( fp, "\n\tFichero ANA :\t%s.ANA\t< relaciones internas >\n",l.id) ;
for( i=0 ; i<80 ; i++ )
        fprintf( fp, "=" ) ;
fprintf( fp, "\n\n") ;

a12( fp ) ;
```

```
fprintf( fp, "\n") ;
for( i=0 ; i<80 ; i++ )
        fprintf( fp, "=" ) ;
fprintf( fp, "\n\nCajas simples :\n") ;

for( i=0 ; i < l.ncajis ; i++ ) {
        fprintf( fp, "\n\tCaja simple %d : ",i ) ;
        for( j=24 ; j<80 ; j++ )
                fprintf( fp, "-" ) ;

        a13( fp, (l.cs+i) ) ;
        }

fprintf( fp, "\nCajas compuestas :\n") ;

for( i=0 ; i < l.ncajic ; i++ ) {
        fprintf( fp, "\n\tCaja compuesta %d : ",i ) ;
        for( j=24 ; j<80 ; j++ )
                fprintf( fp, "-" ) ;

        a14( fp, (l.cc+i) ) ;
        }

fprintf( fp, "\n") ;
for( i=0 ; i<80 ; i++ )
        fprintf( fp, "=" ) ;
fprintf( fp, "\n\nBloques if :\n") ;

for( i=0 ; i < l.nblif ; i++ ) {
        fprintf( fp, "\n\tBloque if %d : ",i ) ;
        for( j=24 ; j<80 ; j++ )
                fprintf( fp, "-" ) ;

        a15( fp, (l.bif+i) ) ;
        }

fprintf( fp, "\n") ;
for( i=0 ; i<80 ; i++ )
        fprintf( fp, "-" ) ;
fprintf( fp, "\n\nBucles :\n") ;

for( i=0 ; i < l.nbuc ; i++ ) {
        fprintf( fp, "\n\tBucle %d : ",i ) ;
        for( j=24 ; j<80 ; j++ )
                fprintf( fp, "-" ) ;

        a16( fp, (l.buc+i) ) ;
        }

fprintf( fp, "\n") ;
for( i=0 ; i<80 ; i++ )
        fprintf( fp, "-" ) ;
fprintf( fp, "\n\nRecursiones :\n\n") ;

for( i=0 ; i < l.nrecur ; i++ ) {
        fprintf( fp, "\tRecurs. %d : ",i ) ;

        a17( fp, (l.rec+i) ) ;
        }
```

```
fprintf( fp, "\n") ;
for( i=0 ; i<80 ; i++ )
        fprintf( fp, "-" ) ;
fprintf( fp, "\n\nInicializaciones :\n") ;

for( i=0 ; i < l.ninic ; i++ ) {
        fprintf( fp, "\n\tInicia. %d : ",i ) ;

        a18( fp, (l.ini+i) ) ;
        }

fprintf( fp, "\n\n") ;
for( i=0 ; i<80 ; i++ )
        fprintf( fp, "=" ) ;

fclose( fp ) ;

}


/*-------------------------------------------------------------------------*/
/*     A11          ESCRITURA DE CAJA CABECERA                             */
/*-------------------------------------------------------------------------*/

a11( fp )

FILE                *fp       ;

{

int                 i         ;
int                 j         ;
struct entrad       *sal      ;
struct salida       *ent      ;


fprintf( fp, "( Id ) \"%s\"\t", l.tree.id ) ;

fprintf( fp, "( Ncadna ) %d\t", l.tree.ncadna ) ;

for( i=0 ; i < l.tree.ncadna ; i++ )
        fprintf( fp, "( %d ) \"%s\"\t", i, *(l.tree.name+i) ) ;

fprintf( fp, "\n( Pohor ) %d\t", l.tree.pohor ) ;
fprintf( fp, "( Pover ) %d\t", l.tree.pover ) ;
fprintf( fp, "( Nentra ) %d\t", l.tree.nentra ) ;
fprintf( fp, "( Nsalid ) %d\t", l.tree.nsalid ) ;

fprintf( fp, "\n\nEnt :\n" ) ;

for( i=0 ; i < l.tree.nentra ; i++ ) {

        ent = (l.tree.ent+i) ;

        fprintf( fp, "\tEnt %d : ( Nam ) \"%s\"\t( Norig ) %d\n", i, ent->nam, ent->norig ) ;

        for( j=0 ; j < ent->norig ; j++ )
                fprintf( fp, "\t\tOrigen %d : ( Nom ) \"%s\"\t( Tipo.Ori ) \"%c.%s\"\n", j, *(ent->nom+j),
*(ent->tipo+j), *(ent->ori+j) ) ;
        }
```

```
fprintf( fp, "\nSal :\n" ) ;

for( i=0 ; i < l.tree.nsalid ; i++ ) {

        sal = (l.tree.sal+i) ;

        fprintf( fp, "\tSal %d : ( Nam ) \"%s\"\t( Ndest ) %d\n", i, sal->nam, sal->ndest ) ;

        for( j=0 ; j < sal->ndest ; j++ )
                fprintf( fp, "\t\tDestin %d : ( Nom ) \"%s\"\t( Tipo.Des ) \"%c.%s\"\n", j, *(sal->nom+j),
*(sal->tipo+j), *(sal->des+j) ) ;
        }


}


/*-------------------------------------------------------------------------*/
/*    A12          ESCRITURA DE RELACIONES INTERNAS                        */
/*-------------------------------------------------------------------------*/

a12( fp )

FILE            *fp     ;

{

int             i       ;
int             j       ;
struct entrad   *ent    ;
struct salida   *sal    ;


fprintf( fp, "( Id ) \"%s\"\t", l.id ) ;

fprintf( fp, "( Name ) \"%s\"\t", l.name ) ;

fprintf( fp, "\n( Nentra ) %d\t", l.nentra ) ;
fprintf( fp, "( Nsalid ) %d\t", l.nsalid ) ;
fprintf( fp, "( Ncajis ) %d\t", l.ncajis ) ;
fprintf( fp, "( Ncajic ) %d\t", l.ncajic ) ;

fprintf( fp, "\n( Nblif ) %d\t", l.nblif  ) ;
fprintf( fp, "( Nbuc ) %d\t", l.nbuc  ) ;
fprintf( fp, "( Nrecur ) %d\t", l.nrecur ) ;
fprintf( fp, "( Ninic ) %d\t", l.ninic  ) ;


fprintf( fp, "\n\nEnt :\n" ) ;

for( i=0 ; i < l.nentra ; i++ ) {

        ent = (l.ent+i) ;

        fprintf( fp, "\tEnt %d : ( Nam ) \"%s\"\t( Ndest ) %d\n", i, ent->nam, ent->ndest ) ;

        for( j=0 ; j < ent->ndest ; j++ )
                fprintf( fp, "\t\tDestin %d : ( Nom ) \"%s\"\t( Tipo.Des ) \"%c.%s\"\n", j, *(ent->nom+j),
*(ent->tipo+j), *(ent->des+j) ) ;
        }
```

```
fprintf( fp, "\nSal :\n" ) ;

for( i=0 ; i < l.nsalid ; i++ ) {

        sal = (l.sal+i) ;

        fprintf( fp, "\tSal %d : ( Nam ) \"%s\"\t( Norig ) %d\n", i, sal->nam, sal->norig ) ;

        for( j=0 ; j < sal->norig ; j++ )
                fprintf( fp, "\t\tOrigen %d : ( Nom ) \"%s\"\t( Tipo.Ori ) \"%c.%s\"\n", j, *(sal->nom+j),
*(sal->tipo+j), *(sal->ori+j) ) ;
        }

}


/*-------------------------------------------------------------------*/
/*     A13          ESCRITURA DE CAJA SIMPLE                         */
/*-------------------------------------------------------------------*/

a13( fp, cs )

FILE                *fp     ;
struct cajita   *cs     ;

{

int                 i       ;
int                 j       ;
struct entrad       *sal    ;
struct salida       *ent    ;


fprintf( fp, "\n\n\t\t( Id ) \"%s\"\t", cs->id ) ;

fprintf( fp, "( Ncadna ) %d\t", cs->ncadna ) ;

for( i=0 ; i < cs->ncadna ; i++ )
        fprintf( fp, "( %d ) \"%s\"\t", i, *(cs->name+i) ) ;

fprintf( fp, "\n\t\t( Pohor ) %d\t", cs->pohor ) ;
fprintf( fp, "( Pover ) %d\t", cs->pover ) ;
fprintf( fp, "( Nentra ) %d\t", cs->nentra ) ;
fprintf( fp, "( Nsalid ) %d\t", cs->nsalid ) ;

fprintf( fp, "\n\n\t\tEnt :\n" ) ;

for( i=0 ; i < cs->nentra ; i++ ) {

        ent = (cs->ent+i) ;

        fprintf( fp, "\t\t\tEnt %d : ( Nam ) \"%s\"\t( Norig ) %d\n", i, ent->nam, ent->norig ) ;

        for( j=0 ; j < ent->norig ; j++ )
                fprintf( fp, "\t\t\t\tOrigen %d : ( Nom ) \"%s\"\t( Tipo.Ori ) \"%c.%s\"\n", j, *(ent-
>nom+j), *(ent->tipo+j), *(ent->ori+j) ) ;
        }

fprintf( fp, "\n\t\tSal :\n" ) ;

for( i=0 ; i < cs->nsalid ; i++ ) {
```

```
        sal = (cs->sal+i) ;

        fprintf( fp, "\t\t\tSal %d : ( Nam ) \"%s\"\t( Ndest ) %d\n", i, sal->nam, sal->ndest ) ;

        for( j=0 ; j < sal->ndest ; j++ )
                fprintf( fp, "\t\t\t\tDestin %d : ( Nom ) \"%s\"\t( Tipo.Des ) \"%c.%s\"\n", j, *(sal-
>nom+j), *(sal->tipo+j), *(sal->des+j) ) ;
        }

}

/*-------------------------------------------------------------------*/
/*    A14          ESCRITURA DE CAJA COMPUESTA                      */
/*-------------------------------------------------------------------*/

a14( fp, cc )

FILE            *fp     ;
struct cajita   *cc ;

{

int             i       ;
int             j       ;
struct entrad   *sal    ;
struct salida   *ent    ;


fprintf( fp, "\n\n\t\t( Id ) \"%s\"\t", cc->id ) ;

fprintf( fp, "( Ncadna ) %d\t", cc->ncadna ) ;

for( i=0 ; i < cc->ncadna ; i++ )
        fprintf( fp, "( %d ) \"%s\"\t", i, *(cc->name+i) ) ;

fprintf( fp, "\n\t\t( Pohor ) %d\t", cc->pohor ) ;
fprintf( fp, "( Pover ) %d\t", cc->pover ) ;
fprintf( fp, "( Nentra ) %d\t", cc->nentra ) ;
fprintf( fp, "( Nsalid ) %d\t", cc->nsalid ) ;

fprintf( fp, "\n\n\t\tEnt :\n" ) ;

for( i=0 ; i < cc->nentra ; i++ ) {

        ent = (cc->ent+i) ;

        fprintf( fp, "\t\t\tEnt %d : ( Nam ) \"%s\"\t( Norig ) %d\n", i, ent->nam, ent->norig ) ;

        for( j=0 ; j < ent->norig ; j++ )
                fprintf( fp, "\t\t\t\tOrigen %d : ( Nom ) \"%s\"\t( Tipo.Ori ) \"%c.%s\"\n", j, *(ent-
>nom+j), *(ent->tipo+j), *(ent->ori+j) ) ;
        }

fprintf( fp, "\n\t\tSal :\n" ) ;

for( i=0 ; i < cc->nsalid ; i++ ) {

        sal = (cc->sal+i) ;
```

```
        fprintf( fp, "\t\t\tSal %d : ( Nam ) \"%s\"\t( Ndest ) %d\n", i, sal->nam, sal->ndest ) ;

        for( j=0 ; j < sal->ndest ; j++ )
                fprintf( fp, "\t\t\t\tDestin %d : ( Nom ) \"%s\"\t( Tipo.Des ) \"%c.%s\"\n", j, *(sal-
>nom+j), *(sal->tipo+j), *(sal->des+j) ) ;
        }

}

/*-------------------------------------------------------------------*/
/*    A15          ESCRITURA DE BLOQUES IF                          */
/*-------------------------------------------------------------------*/

a15( fp, bif )

FILE            *fp     ;
struct blif     *bif    ;


{

int             i       ;
int             j       ;
struct macro    *mac    ;


fprintf( fp, "\n\n\t\t( HueH1 ) %d\t", bif->hueh1 ) ;
fprintf( fp, "( BifPa ) %d\t", bif->bifpa ) ;
fprintf( fp, "( MacMa ) %d\t", bif->macma ) ;
fprintf( fp, "\n\t\t( NivAni ) %d\t", bif->nivani ) ;
fprintf( fp, "( Nmac ) %d\t", bif->nmac ) ;
fprintf( fp, "( Norig ) %d\n", bif->oc.norig ) ;

for( i=0 ; i < bif->nmac ; i++ ) {
        mac = (bif->mac+i) ;
        fprintf( fp, "\n\t\tMac %d :\t( Con ) \"%s\"\t( NivCon ) %d\t( HueH1 ) %d\t\n\t\t\t\t( Ncaj ) %d\t", i,
mac->con, mac->nivcon, mac->hueh1, mac->ncaj ) ;
        for( j=0 ; j < mac->ncaj ; j++ )
                fprintf( fp, "\n\t\t\t\t\t( %d ) \"%s\"\t", j, *(mac->caj+j) ) ;
        }

fprintf( fp, "\n\n" ) ;
for( i=0 ; i < bif->oc.norig ; i++ )
        fprintf( fp, "\t\tOrigen %d : ( Nom ) \"%s\"\t( Tipo.Ori ) \"%c.%s\"\n", i, *(bif->oc.nom+i), *(bif-
>oc.tipo+i), *(bif->oc.ori+i) ) ;

}

/*-------------------------------------------------------------------*/
/*    A16          ESCRITURA DE BUCLES                              */
/*-------------------------------------------------------------------*/

a16( fp, buc )

FILE            *fp     ;
struct bucli    *buc    ;

{
```

```
int                 i       ;

fprintf( fp, "\n\n\t\t( NCaj ) %d\t", buc->ncaj ) ;
for( i=0 ; i < buc->ncaj ; i++ )
        fprintf( fp, "\n\t\t\t( %d ) \"%s\"\t", i, *(buc->caj+i) ) ;

fprintf( fp, "\n\t\t( Texaux ) \"%s\"\t( Texcon ) \"%s\"\n", buc->texaux, buc->texcon ) ;

fprintf( fp, "\t\t( NOrCon ) %d\t", buc->oc.norig ) ;
fprintf( fp, "( NOrBuc ) %d\t", buc->ob.norig ) ;
fprintf( fp, "( NExBuc ) %d\t", buc->eb.norig ) ;

fprintf( fp, "\n\t\t( NivCon ) %d\t", buc->nivcon ) ;
fprintf( fp, "( NivAni ) %d\t", buc->nivani ) ;

fprintf( fp, "( Tipo.ExCon ) \"%c.%s\"\n", buc->tipo, buc->excon ) ;

fprintf( fp, "\n\t\tOrig. Con. :\n" ) ;

for( i=0 ; i < buc->oc.norig ; i++ )
        fprintf( fp, "\t\t\tOrigen %d : ( Nom ) \"%s\"\t( Tipo.Ori ) \"%c.%s\"\n", i, *(buc->oc.nom+i),
*(buc->oc.tipo+i), *(buc->oc.ori+i) ) ;

fprintf( fp, "\n\t\tOrig. Buc. :\n" ) ;

for( i=0 ; i < buc->ob.norig ; i++ )
        fprintf( fp, "\t\t\tOrigen %d : ( Nom ) \"%s\"\t( Tipo.Ori ) \"%c.%s\"\n", i, *(buc->ob.nom+i),
*(buc->ob.tipo+i), *(buc->ob.ori+i) ) ;

fprintf( fp, "\n\t\tExtr. Buc. :\n" ) ;

for( i=0 ; i < buc->eb.norig ; i++ )
        fprintf( fp, "\t\t\tExtrem %d : ( Nom ) \"%s\"\t( Tipo.Ori ) \"%c.%s\"\n", i, *(buc->eb.nom+i),
*(buc->eb.tipo+i), *(buc->eb.ori+i) ) ;

}

/*-------------------------------------------------------------------*/
/*    A17          ESCRITURA DE RECURSIONES                          */
/*-------------------------------------------------------------------*/

a17( fp, rec )

FILE                *fp     ;
struct cursi        *rec    ;

{

fprintf( fp, "\t( Caja ) \"%s\"", rec->caja ) ;

fprintf( fp, "\t( Tex ) \"%s\"\n", rec->tex ) ;

}

/*-------------------------------------------------------------------*/
/*    A18          ESCRITURA DE INICIALIZACIONES                     */
/*-------------------------------------------------------------------*/
```

```
a18( fp, ini )

FILE            *fp     ;
struct inici    *ini    ;

{

fprintf( fp, "\t( Caja ) \"%s\"", ini->caja ) ;

fprintf( fp, "\t( Inp ) \"%s\"", ini->inp ) ;

fprintf( fp, "\t( Val ) \"%s\"", ini->val ) ;

fprintf( fp, "\t( AnCa ) \"%s\"\t", ini->anca ) ;

}
```

```
#include "head.h"
#include <stdlib.h>
#include <graphics.h>
#include <math.h>

extern struct dibujo        d       ;
extern struct grafic        g       ;

/*-------------------------------------------------------------------*/
/*    A3                   G A UNA ESCALA DADA                       */
/*-------------------------------------------------------------------*/

a3( una, une, facy, flag1, pm, panOimp1 )

struct unidad       *una    ;
struct unidad       *une    ;
int                 facy[ ] ;
int                 flag1   ;
int                 pm[ ]   ;
int                 panOimp1;

{

int                 *lxca   ;
int                 *lyca   ;
int                 *xx     ;
int                 *yy     ;
int                 nx      ;
int                 ny      ;
int                 i       ;
struct unidad       *un     ;                    /*static int  tem=0  ;*/
int                 *mem_int(int,char * );

                                                                  /*tem++ ;*/

if( flag1 == 0 ) {
        un = una ;
        facy[0] = 1 ;
        facy[1] = 1 ;
        }
else    {
        un = une ;
        }

pm[0] = d.dr.tw * un->tw + d.dr.hu * un->hu ;
pm[1] = d.dr.th * un->th + d.dr.vu * un->vu ;

nx = d.dc.ncah ;
ny = d.dc.ncav ;                          /*printf("\n nx = %d  ny = %d",nx,ny);*/

if( ny > 0 ) {
        lxca = mem_int( nx, "a3(lxca)" ) ;
        lyca = mem_int( ny, "a3(lyca)" ) ;
        }

xx = mem_int( 2*(nx+1), "a3(xx)" ) ;
yy = mem_int( 2*(ny+1), "a3(yy)" ) ;

/*if(tem==2){closegraph();exit(1);}*/
a31( &(d.dc), d.dh.nchmar, un, lxca, lyca, xx, yy ) ;          /*if(tem==2){closegraph();exit(1);}*/

a32( &(d.dc), &(d.dh), un, lxca, lyca, xx, yy ) ;
```

```
a33( &(d.re.te), un, xx, yy ) ;

a34( d.re.nrect, d.re.re, d.re.li.nling, d.re.li.lg, un, xx, yy ) ;

a35( d.re.li.nlinf, d.re.li.lf, d.re.narc, d.re.ar, d.re.nfle, d.re.fl, un, xx, yy, panOimp1 ) ;

a36( d.re.nmar1, d.re.ma1, d.re.nmar2, d.re.ma2, un, xx, yy ) ;

if( ny > 0 )  {
        free( lxca ) ;
        free( lyca ) ;
        }
free( xx ) ;
free( yy ) ;

}


/*-------------------------------------------------------------------*/
/*    A31   ANCHURAS, ALTURAS Y MARGENES                             */
/*-------------------------------------------------------------------*/

a31( dc, nchmar, un, lxca, lyca, xx, yy )

struct dimcaj      *dc     ;
int                nchmar  ;
struct unidad      *un     ;
int                lxca[ ]  ;
int                lyca[ ]  ;
int                xx[ ]    ;
int                yy[ ]    ;

{

int                i       ;                          /*static int  tem=0  ;*/

                                                                  /*tem++ ;*/
for( i=0 ; i < dc->ncav ; i++ )
        lyca[i] = (dc->cav+i)->nvu * un->vu + (dc->cav+i)->ncad * un->th ;

/*if(tem==2){closegraph();exit(1);} printf("\n dc->ncah = %d",dc->ncah);*/
for( i=0 ; i < dc->ncah ; i++ ) {                          /*printf("\n i= %d",i ) ;*/
        lxca[i]  = un->tw * ( (dc->cah+i)->nchi + (dc->cah+i)->ncho ) ;
        lxca[i] += ( un->tw * (dc->cah+i)->nchn + un->hu * ( 4 * 2 + 2 * 1 ) ) ;
        }                                          /*getch();if(tem==2){closegraph();exit(1);}*/

xx[0] = un->tw * nchmar + un->hu * ( 1 + 1 ) ;
yy[0] = un->th * 2      + un->vu * ( 1 + 2 + 2 ) ;

}


/*-------------------------------------------------------------------*/
/*    A32   AJUSTE DE SISTEMAS DE REFERENCIA                         */
/*-------------------------------------------------------------------*/

a32( dc, dh, un, lxca, lyca, xx, yy )

struct dimcaj      *dc     ;
struct dimhue      *dh     ;
```

```
struct unidad       *un     ;
int                 lxca[ ] ;
int                 lyca[ ] ;
int                 xx[ ]   ;
int                 yy[ ]   ;

{

int                 i       ;
int                 j       ;


for( i=0, j=1 ; i < dc->ncah ; i++, j+=2 )  {
        xx[j]   = xx[j-1] + *(dh->nhu+i) * un->hu + *(dh->ntw+i) * un->tw ;
        xx[j+1] = xx[j]   + lxca[i] ;
        }
xx[j] = xx[j-1] + *(dh->nhu+i) * un->hu + *(dh->ntw+i) * un->tw ;

for( i=0, j=1 ; i < dc->ncav ; i++, j+=2 )  {
        yy[j]   = yy[j-1] + *(dh->nvu+i) * un->vu + *(dh->nth+i) * un->th ;
        yy[j+1] = yy[j]   + lyca[i] ;
        }
yy[j] = yy[j-1] + *(dh->nvu+i) * un->vu + *(dh->nth+i) * un->th ;

}


/*-------------------------------------------------------------------*/
/*    A33                              TEXTOS EN G                   */
/*-------------------------------------------------------------------*/

a33( te, un, xx, yy )

struct texdib       *te     ;
struct unidad       *un     ;
int                 xx[ ]   ;
int                 yy[ ]   ;

{

int                 srh     ;
int                 srv     ;
int                 i       ;
struct texele       *tx     ;
struct teg          *tg     ;
char                *mem_char(int,char * );
struct teg          *mem_teg(int,char * );


if( te->ntdat >0 )
        g.tx.tgd = mem_teg( te->ntdat, "a33(g.tx.tgd)" ) ;
g.tx.ntd = te->ntdat ;

for( i=0 ; i < te->ntdat ; i++ )    {

        tg = (g.tx.tgd+i) ;

        tx = (te->td+i) ;
        srh = tx->ch.sr ;
        srv = tx->cv.sr ;
```

```
        tg->x = xx[srh] + tx->ch.hu * un->hu + tx->ch.tw * un->tw ;
        tg->y = yy[srv] + tx->cv.vu * un->vu + tx->cv.th * un->th ;

        (g.tx.tgd+i)->s = mem_char( ( strlen( tx->s ) + 1 ), "a33(g.tx.tgd->s)" ) ;

        strcpy( tg->s, tx->s ) ;
        }

if( te->ntcaj >0 )
        g.tx.tgc = mem_teg( te->ntcaj, "a33(g.tx.tgc)" ) ;
g.tx.ntcj = te->ntcaj ;

for( i=0 ; i < te->ntcaj ; i++ )    {

        tg = (g.tx.tgc+i) ;

        tx = (te->tc+i) ;
        srh = tx->ch.sr ;
        srv = tx->cv.sr ;

        tg->x = xx[srh] + tx->ch.hu * un->hu + ( tx->ch.tw * un->tw ) / 2. ;
        tg->y = yy[srv] + tx->cv.vu * un->vu + tx->cv.th * un->th ;

        (g.tx.tgc+i)->s = mem_char( ( strlen( tx->s ) + 1 ), "a33(g.tx.ygc->s)" ) ;

        strcpy( tg->s, tx->s ) ;
        }

srh = te->tt.ch.sr ;
srv = te->tt.cv.sr ;

g.tx.tgt.x = xx[srh] + te->tt.ch.hu * un->hu + te->tt.ch.tw * un->tw ;
g.tx.tgt.y = yy[srv] + te->tt.cv.vu * un->vu + te->tt.cv.th * un->th ;

g.tx.tgt.s = mem_char( ( strlen( te->tt.s ) + 1 ), "a33(g.tx.tgt.s)" ) ;

strcpy( g.tx.tgt.s, te->tt.s ) ;

}


/*-------------------------------------------------------------------*/
/*      A34     RECTANGULOS Y LINEAS GRUESAS EN G                    */
/*-------------------------------------------------------------------*/

a34( n, re, m, lg, un, xx, yy )

int             n       ;
struct rectan   *re     ;
int             m       ;
struct lingru   *lg     ;
struct unidad   *un     ;
int             xx[ ]   ;
int             yy[ ]   ;

{

int             srh     ;
int             srv     ;
int             i       ;
int             j       ;
```

```
struct rectan   *rc ;
struct lingru       *gg     ;
struct rect         *rt     ;
struct rect         *rtt    ;
struct line         *lf1    ;
struct line         *lf2    ;
struct rect         *mem_rect(int,char * );
struct line         *mem_line(int,char * );


g.re = mem_rect( 2*n, "a34(g.re)" ) ;
g.nrect = 2*n ;

for( i=0,j=0 ; i < n ; i++,j+=2 )    {

        rt = (g.re+j) ;
        rtt = (g.re+j+1) ;
        rc = (re+i) ;
        srh = rc->ch.sr ;
        srv = rc->cv.sr ;

        rt->pih = xx[srh] + rc->ch.hu * un->hu + rc->ch.tw * un->tw ;
        rt->piv = yy[srv] + rc->cv.vu * un->vu + rc->cv.th * un->th ;

        rt->pfh = rt->pih + rc->dr.hu * un->hu + rc->dr.tw * un->tw ;
        rt->pfv = rt->piv + rc->dr.vu * un->vu + rc->dr.th * un->th ;

        rtt->pih = rt->pih + 1 ;
        rtt->piv = rt->piv + 1 ;
        rtt->pfh = rt->pfh - 1 ;
        rtt->pfv = rt->pfv - 1 ;
        }

g.li = mem_line( 2*m, "a34(g.li)" ) ;
g.nline = 2*m ;

for( i=0,j=0 ; i < m ; i++,j+=2 )    {

        lf1 = (g.li+j) ;
        lf2 = (g.li+j+1) ;
        gg = (lg+i) ;
        srh = gg->ch[0].sr ;
        srv = gg->cv[0].sr ;

        lf1->pih = xx[srh] + gg->ch[0].hu * un->hu + gg->ch[0].tw * un->tw ;
        lf1->piv = yy[srv] + gg->cv[0].vu * un->vu + gg->cv[0].th * un->th ;

        lf1->pfh = xx[srh] + gg->ch[1].hu * un->hu + gg->ch[1].tw * un->tw ;
        lf1->pfv = yy[srv] + gg->cv[1].vu * un->vu + gg->cv[1].th * un->th ;

        lf2->pih = lf1->pih ;
        lf2->piv = lf1->piv - 1 ;
        lf2->pfh = lf1->pfh ;
        lf2->pfv = lf1->pfv - 1 ;
        }
}

/*-------------------------------------------------------------------*/
/*    A35          FLECHAS, ARCOS Y LINEAS FINAS EN G                */
```

```
/*-------------------------------------------------------------------------*/

a35( nl, lf, na, ar, nf, fl, un, xx, yy, panOimp1 )

int             nl      ;
struct linfin   *lf     ;
int             na      ;
struct arcrec   *ar     ;
int             nf      ;
struct flecha   *fl     ;
struct unidad   *un     ;
int             xx[ ]   ;
int             yy[ ]   ;
int             panOimp1;

{

int             srh     ;
int             srv     ;
int             i       ;
int             x[2]    ;
int             y[2]    ;
int             rad     ;
long            rady    ;
int             xasp    ;
int             yasp    ;


if( panOimp1 == 0 ) {
        a351( nf, fl, un, xx, yy ) ;

        getaspectratio( &xasp, &yasp ) ;
        rad = un->th ;
        rady = (long)rad * (long)xasp / (long)yasp ;
        }
else    {
        a354( nf, fl, un, xx, yy ) ;

        rad = 2 * un->tw ;
        rady = un->th ;
        }

a352( na, ar, un, xx, yy, rad, (int)rady, panOimp1 ) ;

a353( nl, lf, un, xx, yy, rad, (int)rady ) ;

}


/*-------------------------------------------------------------------------*/
/*    A351        FLECHAS          EN G                                    */
/*-------------------------------------------------------------------------*/

a351( nf, fl, un, xx, yy )

int             nf      ;
struct flecha   *fl     ;
struct unidad   *un     ;
int             xx[ ]   ;
int             yy[ ]   ;
```

```
{

int                 srh     ;
int                 srv     ;
int                 i       ;
int                 lon     ;
int                 gro     ;
struct flecha       *ff     ;
struct fle *fg      ;
struct fle *mem_fle(int,char * );


lon = 2 * min( un->hu, un->vu ) ;
gro = 2 * lon / 3 ;

if( nf > 0 )
        g.fl = mem_fle( nf, "a351(g.fl)" ) ;
g.nfle = nf ;

for( i=0 ; i < nf ; i++ )   {

        fg = (g.fl+i) ;
        ff = (fl+i) ;
        srh = (ff->ch)->sr ;
        srv = (ff->cv)->sr ;

        fg->x[0] = xx[srh] + (ff->ch)->hu * un->hu + (ff->ch)->tw * un->tw ;
        fg->x[1] = yy[srv] + (ff->cv)->vu * un->vu + (ff->cv)->th * un->th ;

        switch( ff->npimed )  {

                case 0 :
                        fg->x[2] = fg->x[0] - lon ;
                        fg->x[3] = fg->x[1] - gro ;
                        fg->x[4] = fg->x[2] ;
                        fg->x[5] = fg->x[1] + gro ;
                        break ;
                case 3 :
                        fg->x[2] = fg->x[0] + gro ;
                        fg->x[3] = fg->x[1] - lon ;
                        fg->x[4] = fg->x[0] - gro ;
                        fg->x[5] = fg->x[3] ;
                        break ;
                }

        }

}

/*-------------------------------------------------------------*/
/*    A352                 ARCOS EN G                          */
/*-------------------------------------------------------------*/

a352( na, ar, un, xx, yy, rad, rady, panOimp1 )

int                 na      ;
struct arcrec       *ar     ;
struct unidad       *un     ;
int                 xx[ ]   ;
```

```
int             yy[ ]    ;
int             rad      ;
int             rady     ;
int             panOimp1;

{

int             srh      ;
int             srv      ;
int             i        ;
int             x        ;
int             y        ;
struct arcrec   *aa      ;
struct arc      *ac      ;
struct arc      *mem_arc(int,char * );


if( na >0 )
        g.ar = mem_arc( na, "a352(g.ar)" ) ;
g.narc = na ;

for( i=0 ; i < na ; i++ ) {

        ac = (g.ar+i) ;
        aa = (ar+i) ;
        srh = (aa->ch)->sr ;
        srv = (aa->cv)->sr ;

        x = xx[srh] + (aa->ch)->hu * un->hu + (aa->ch)->tw * un->tw ;
        y = yy[srv] + (aa->cv)->vu * un->vu + (aa->cv)->th * un->th ;

        switch( aa->cuadr ) {

                case 1 :
                        ac->x = x - rad ;
                        ac->y = y + rady ;
                        ac->iz = 0 ;
                        if( panOimp1 == 0 )
                                ac->fz = 90 ;
                        else
                                ac->fz = rady ;
                        ac->ra = rad ;
                        break ;
                case 2 :
                        ac->x = x + rad ;
                        ac->y = y + rady ;
                        ac->iz = 90 ;
                        if( panOimp1 == 0 )
                                ac->fz = 180 ;
                        else
                                ac->fz = rady ;
                        ac->ra = rad ;
                        break ;
                case 3 :
                        ac->x = x + rad ;
                        ac->y = y - rady ;
                        ac->iz = 180 ;
                        if( panOimp1 == 0 )
                                ac->fz = 270 ;
                        else
                                ac->fz = rady ;
```

```
                        ac->ra = rad ;
                        break ;
                case 4 :
                        ac->x = x - rad ;
                        ac->y = y - rady ;
                        ac->iz = 270 ;
                        if( panOimp1 == 0 )
                                ac->fz = 360 ;
                        else
                                ac->fz = rady ;
                        ac->ra = rad ;
                        break ;
                }
        }
}

/*-------------------------------------------------------------------*/
/*     A353                LINEAS FINAS EN G                         */
/*-------------------------------------------------------------------*/

a353( nl, lf, un, xx, yy, rad, rady )

int             nl      ;
struct linfin   *lf     ;
struct unidad   *un     ;
int             xx[ ]   ;
int             yy[ ]   ;
int             rad     ;
int             rady    ;

{

int             srh0    ;
int             srv0    ;
int             srh1    ;
int             srv1    ;
int             i       ;
struct linfin   *ll     ;
struct line     *lf0    ;
struct line     *memr_line(struct line * ,int,char * );


if( nl > 0 )
        g.li = memr_line( g.li, ( g.nline + nl ), "a353(g.li)" ) ;

for( i=0 ; i < nl ; i++ )    {

        lf0 = (g.li + g.nline + i ) ;
        ll = (lf+i) ;
        srh0 = (ll->x0)->sr ;
        srh1 = (ll->x1)->sr ;
        srv0 = (ll->y0)->sr ;
        srv1 = (ll->y1)->sr ;

        lf0->pih = xx[srh0] + (ll->x0)->hu * un->hu + (ll->x0)->tw * un->tw ;
        lf0->piv = yy[srv0] + (ll->y0)->vu * un->vu + (ll->y0)->th * un->th ;

        lf0->pfh = xx[srh1] + (ll->x1)->hu * un->hu + (ll->x1)->tw * un->tw ;
        lf0->pfv = yy[srv1] + (ll->y1)->vu * un->vu + (ll->y1)->th * un->th ;
```

```
            lf0->pih += ( ll->modx[0] * rad ) ;
            lf0->pfh -= ( ll->modx[1] * rad ) ;
            lf0->piv += ( ll->mody[0] * rady ) ;
            lf0->pfv -= ( ll->mody[1] * rady ) ;

            }

    g.nline += nl ;

    }


    /*-------------------------------------------------------------------*/
    /*    A354        FLECHAS          EN G PARA LA IMPRESION            */
    /*-------------------------------------------------------------------*/

    a354( nf, fl, un, xx, yy )

    int                 nf      ;
    struct flecha       *fl     ;
    struct unidad       *un     ;
    int                 xx[ ]   ;
    int                 yy[ ]   ;

    {

    int                 srh     ;
    int                 srv     ;
    int                 i       ;
    int                 lon     ;
    int                 gro     ;
    struct flecha       *ff     ;
    struct fle *fg      ;
    struct fle *mem_fle(int,char * );


    if( nf > 0 )
            g.fl = mem_fle( nf, "a354(g.fl)" ) ;
    g.nfle = nf ;

    for( i=0 ; i < nf ; i++ )   {

            fg = (g.fl+i) ;
            ff = (fl+i) ;
            srh = (ff->ch)->sr ;
            srv = (ff->cv)->sr ;

            fg->x[0] = xx[srh] + (ff->ch)->hu * un->hu + (ff->ch)->tw * un->tw ;
            fg->x[1] = yy[srv] + (ff->cv)->vu * un->vu + (ff->cv)->th * un->th ;

            switch( ff->npimed )  {

                    case 0 :
                            fg->x[2] = fg->x[0] - un->hu * 2 ;
                            fg->x[3] = fg->x[1] - un->vu * 4 / 3 ;
                            fg->x[4] = fg->x[2] ;
                            fg->x[5] = fg->x[1] + un->vu * 4 / 3 ;
                            break ;
                    case 3 :
                            fg->x[2] = fg->x[0] + un->hu ;
                            fg->x[3] = fg->x[1] - un->vu * 3 ;
```

```
                        fg->x[4] = fg->x[0] - un->hu ;
                        fg->x[5] = fg->x[3] ;
                        break ;
                }

        }

}

/*-------------------------------------------------------------------*/
/*    A4                    DIBUJAR EN PANTALLA                    */
/*-------------------------------------------------------------------*/


a4( g, facy, cte, col )

struct grafic       *g      ;
int                 facy[ ] ;
int                 cte[ ]  ;
struct color        col     ;

{

int                 i       ;
int                 x[6]    ;
struct rect         *r      ;
struct line         *ln     ;
struct teg          *t      ;
struct fle *f       ;
struct arc          *a      ;


cleardevice( ) ;
setcolor( col.dibu ) ;
setlinestyle( SOLID_LINE, 0, NORM_WIDTH ) ;

for( i=0 ; i < g->nrect ; i++ ) {

        r = (g->re+i) ;

        rectangle( (r->pih-cte[0]), (r->piv-cte[1]), (r->pfh-cte[0]), (r->pfv-cte[1]) ) ;
        }

for( i=0 ; i <g->nline ; i++ ) {

        ln = (g->li+i) ;

        line( (ln->pih-cte[0]), (ln->piv-cte[1]), (ln->pfh-cte[0]), (ln->pfv-cte[1]) ) ;
        }

settextstyle( SMALL_FONT, HORIZ_DIR, 1 ) ;
settextjustify( CENTER_TEXT, BOTTOM_TEXT ) ;

do      {
        if( facy[0] < 240 )
                break ;
        facy[0] = facy[0]/2 ;
        facy[1] = facy[1]/2 ;
        }
while( 1 ) ;
```

```
setusercharsize( 2 * facy[0], facy[1], 2 * facy[0], facy[1] ) ;

outtextxy( ( g->tx.tgt.x - cte[0] ), ( g->tx.tgt.y - cte[1] ), g->tx.tgt.s ) ;

setusercharsize( facy[0], facy[1], facy[0], facy[1] ) ;

for( i=0 ; i < g->tx.ntcj ; i++ ) {

        t = (g->tx.tgc+i) ;

        outtextxy( ( t->x - cte[0] ), ( t->y - cte[1] ), t->s ) ;
        }

settextjustify( LEFT_TEXT, BOTTOM_TEXT ) ;

for( i=0 ; i < g->tx.ntd ; i++ ) {

        t = (g->tx.tgd+i) ;

        outtextxy( ( t->x - cte[0] ), ( t->y - cte[1] ), t->s ) ;
        }

for( i=0 ; i < g->narc ; i++ ) {

        a = (g->ar+i) ;

        arc( ( a->x - cte[0] ), ( a->y - cte[1] ), a->iz, a->fz, a->ra ) ;
        }

setfillstyle( SOLID_FILL, col.dibu ) ;

for( i=0 ; i < g->nfle ; i++ ) {

        f = (g->fl+i) ;

        x[0] = f->x[0] - cte[0] ;
        x[1] = f->x[1] - cte[1] ;
        x[2] = f->x[2] - cte[0] ;
        x[3] = f->x[3] - cte[1] ;
        x[4] = f->x[4] - cte[0] ;
        x[5] = f->x[5] - cte[1] ;

        fillpoly( 3, x ) ;
        }

}

/*-------------------------------------------------------------------------*/
/*      A5                              CAMBIO DE ESCALA                   */
/*-------------------------------------------------------------------------*/

a5( dr, una, une, un, xymax, facy, col )

struct unidad     *dr     ;
struct unidad     *una    ;
struct unidad     *une    ;
struct unidad     *un     ;
int               xymax[ ];
int               facy[ ]  ;
```

```
struct color        col     ;

{

int                 accion  ;
int                 npxy[2] ;
int                 me2( )  ;
double              x       ;


switch( me2( un, xymax, col, npxy ) ) {

        case -1 :
                return( -1 ) ;
        case 0 :
                une->tw = una->tw ;
                une->hu = una->hu ;
                une->th = una->th ;
                une->vu = una->vu ;
                facy[0] = 1 ;
                facy[1] = 1 ;
                break ;
        case 1 :
                x = 1 ;
                a52( dr, una, une, xymax, facy, x ) ;
                break ;
        case 2 :
                x = sqrt( 2. ) ;
                a52( dr, una, une, xymax, facy, x ) ;
                break ;
        case 3 :
                x = 2 ;
                a52( dr, una, une, xymax, facy, x ) ;
                break ;
        case 4 :
                a51( dr, una, une, xymax, facy, npxy ) ;

        }

}

/*-------------------------------------------------------------------------*/
/*      A51                             ESCALA MANUAL                      */
/*-------------------------------------------------------------------------*/

a51( dr, una, un, xymax, facy, npxy )

struct unidad       *dr     ;
struct unidad       *una    ;
struct unidad       *un     ;
int                 xymax[ ];
int                 facy[ ] ;
int                 npxy[ ] ;

{

double              factor  ;
int                 xmin    ;
int                 ymin    ;
int                 leftx   ;
```

```
int             lefty   ;

xmin = 2 * dr->hu + dr->tw ;
ymin = 2 * dr->vu + dr->th ;

if( npxy[0] == 0 )
        npxy[0] = xymax[0] ;
if( npxy[1] == 0 )
        npxy[1] = xymax[1] ;
if( npxy[0] <= xmin )
        npxy[0] = xmin ;
if( npxy[1] <= ymin )
        npxy[1] = ymin ;

factor = (double) npxy[0] / ( dr->hu * una->hu + dr->tw * una->tw ) ;

if( factor * ( dr->vu * una->vu + dr->th * una->th ) > npxy[1] )
        factor = (double) npxy[1] / ( dr->vu * una->vu + dr->th * una->th ) ;

factor = ( factor > 100.) ? 100. : factor ;
facy[0] = 100 * factor ;
facy[1] = 100 ;

un->tw = una->tw * facy[0] / facy[1] ;
un->th = una->th * facy[0] / facy[1] ;

leftx = npxy[0] - dr->tw * un->tw ;
lefty = npxy[1] - dr->th * un->th ;

un->hu = leftx / dr->hu ;
un->vu = lefty / dr->vu ;

}

/*-------------------------------------------------------------------*/
/*    A52                  ESCALA TIPO DIN                          */
/*-------------------------------------------------------------------*/

a52( dr, una, un, xymax, facy, x )

struct unidad   *dr ;
struct unidad   *una    ;
struct unidad   *un     ;
int             xymax[ ];
int             facy[ ] ;
double          x       ;

{

double          factor  ;
int             leftx   ;
int             lefty   ;


factor = ( xymax[0] * x ) / ( dr->hu * una->hu + dr->tw * una->tw ) ;

if( factor * ( dr->vu * una->vu + dr->th * una->th ) > ( xymax[1] * x ) )
        factor = ( xymax[1] * x ) / ( dr->vu * una->vu + dr->th * una->th ) ;
```

```
facy[0] = 100 * factor ;
facy[1] = 100 ;

un->tw = una->tw * facy[0] / facy[1] ;
un->th = una->th * facy[0] / facy[1] ;

leftx = ( xymax[0] * x ) - dr->tw * un->tw ;
lefty = ( xymax[1] * x ) - dr->th * un->th ;

un->hu = leftx / dr->hu ;
un->vu = lefty / dr->vu ;

}

/*-------------------------------------------------------------------*/
/*      A6                          CAMBIO DE CENTRO                 */
/*-------------------------------------------------------------------*/

a6( xymax, xyo, cte, pm, col )

int             xymax[ ];
int             xyo[ ]   ;
int             cte[ ]   ;
int             pm[ ]    ;
struct color    col      ;

{

int             n = 1    ;
int             a        ;
int             b        ;
int             c        ;
int             d        ;
int             auxm1    ;
int             auxm2    ;
int             np = 10  ;
int             lp = 80  ;
int             xyr[ ] = { 0, 0 } ;
int             xya[ ] = { 0, 0 } ;
void            *buff11  ;
void            *buff12  ;
void            *buff21  ;
void            *buff22  ;
void            *buff3   ;
void            *buff4   ;
union inkey     {
                char    ch[2]    ;
                int     i        ;
                }       cc       ;

setlinestyle( SOLID_LINE, 0, NORM_WIDTH ) ;
setcolor( col.reor ) ;

if( ( cte[0] == 0 ) && ( cte[1] == 0 ) ) {
        xyo[0] = 0 ;
        xyo[1] = 0 ;
        }

do      {
```

```
        a = xyr[1]+lp-1 ;
        b = xyr[1]+xymax[1]-lp+1 ;
        c = xyr[1]+xymax[1] ;
        d = xyr[0]+xymax[0] ;

        if( xyr[0] <= 0 ) {

                if( xyr[0] == 0 ) {

                        if( xyr[1] < 0 ) {

                                if( a >= 0 ) {
                                        if( (buff11 = malloc(imagesize(0,0,0,a))) == NULL )
                                                { closegraph( ) ; perror("error: a6.a:
memoria(malloc): ") ; exit( 1 ) ; }
                                        getimage(0,0,0,a,buff11) ;
                                        }

                                if( (buff12 = malloc(imagesize(0,max(0,b),0,c))) == NULL )
                                        { closegraph( ) ; perror("error: a6.b: memoria(malloc): ") ;
exit( 1 ) ; }
                                getimage(0,max(0,b),0,c,buff12 ) ;
                                }

                        else    {

                                auxm1 = min(xymax[1],a) ;

                                if( (buff11 = malloc(imagesize(0,xyr[1],0,auxm1))) == NULL )
                                        { closegraph( ) ; perror("error: a6.c: memoria(malloc): ") ;
exit( 1 ) ; }
                                getimage(0,xyr[1],0,auxm1,buff11 ) ;

                                if( ( xyr[1] - lp ) < 0 ) {
                                        if( (buff12 = malloc(imagesize(0,b,0,c))) == NULL )
                                                { closegraph( ) ; perror("error: a6.d:
memoria(malloc): ") ; exit( 1 ) ; }
                                        getimage(0,b,0,c,buff12 ) ;
                                        }
                                }
                        }

                if( xyr[1] < 0 ) {

                        if( a >= 0 ) {
                                if( (buff21 = malloc(imagesize(d,0,d,a))) == NULL )
                                        { closegraph( ) ; perror("error: a6.e: memoria(malloc): ") ;
exit( 1 ) ; }
                                getimage(d,0,d,a,buff21 ) ;
                                }

                        auxm1 = max(0,b) ;

                        if( (buff22 = malloc(imagesize(d,auxm1,d,c))) == NULL )
                                { closegraph( ) ; perror("error: a6.f: memoria(malloc): ") ; exit( 1 ) ; }
                        getimage(d,auxm1,d,c,buff22 ) ;
                        }

                else    {
```

```
                        auxm1 = min(xymax[1],a) ;

                        if( (buff21 = malloc(imagesize(d,xyr[1],d,auxm1))) == NULL )
                                { closegraph( ) ; perror("error: a6.g: memoria(malloc): ") ; exit( 1 ) ; }
                        getimage(d,xyr[1],d,auxm1,buff21 ) ;

                        if( ( xyr[1] - lp ) < 0 ) {
                                auxm1 = min(xymax[1],c) ;
                                if( (buff22 = malloc(imagesize(d,b,d,auxm1))) == NULL )
                                        { closegraph( ) ; perror("error: a6.h: memoria(malloc): ") ;
exit( 1 ) ; }
                                getimage(d,b,d,auxm1,buff22 ) ;
                                }
                        }
                }

        else    {

                if( xyr[1] < 0 ) {

                        if( a >= 0 ) {
                                if( (buff11 = malloc(imagesize(xyr[0],0,xyr[0],a))) == NULL )
                                        { closegraph( ) ; perror("error: a6.i: memoria(malloc): ") ; exit(
1 ) ; }
                                getimage(xyr[0],0,xyr[0],a,buff11) ;
                                }

                        auxm1 = max(0,b) ;
                        if( (buff12 = malloc(imagesize(xyr[0],auxm1,xyr[0],c))) == NULL )
                                { closegraph( ) ; perror("error: a6.j: memoria(malloc): ") ; exit( 1 ) ; }
                        getimage(xyr[0],auxm1,xyr[0],c,buff12) ;
                        }

                else    {

                        auxm1 = min(xymax[1],a) ;
                        if( (buff11 = malloc(imagesize(xyr[0],xyr[1],xyr[0],auxm1))) == NULL )
                                { closegraph( ) ; perror("error: a6.k: memoria(malloc): ") ; exit( 1 ) ; }
                        getimage(xyr[0],xyr[1],xyr[0],auxm1,buff11) ;

                        if( ( xyr[1] - lp ) < 0 ) {
                                auxm1 = min(xymax[1],c) ;
                                if( (buff12 = malloc(imagesize(xyr[0],b,xyr[0],auxm1))) == NULL )
                                        { closegraph( ) ; perror("error: a6.l: memoria(malloc): ") ; exit(
1 ) ; }
                                getimage(xyr[0],b,xyr[0],auxm1,buff12) ;
                                }
                        }
                }

        auxm1 = min(xymax[0],d) ;
        auxm2 = max(0,xyr[0]) ;

        if( xyr[1] <= 0 ) {

                if( xyr[1] == 0 ) {
                        if( (buff3 = malloc(imagesize(auxm2,0,auxm1,0))) == NULL )
                                { closegraph( ) ; perror("error: a6.m: memoria(malloc): ") ; exit( 1 ) ; }
                        getimage(auxm2,0,auxm1,0,buff3) ;
                        }
```

```
        if( (buff4 = malloc(imagesize(auxm2,c,auxm1,c))) == NULL )
                { closegraph( ) ; perror("error: a6.n: memoria(malloc): ") ; exit( 1 ) ; }
        getimage(auxm2,c,auxm1,c,buff4) ;
        }

else    {
        if( (buff3 = malloc(imagesize(auxm2,xyr[1],auxm1,xyr[1]))) == NULL )
                { closegraph( ) ; perror("error: a6.o: memoria(malloc): ") ; exit( 1 ) ; }
        getimage(auxm2,xyr[1],auxm1,xyr[1],buff3) ;
        }

a61( lp, xyr, xymax ) ;

do      {
        while( !bioskey( 1 ) ) ;
        cc.i = bioskey( 0 ) ;

        if( cc.ch[0] ) {
                if( cc.ch[0] == '\r' ) {
                        cte[0] = xyr[0] + xyo[0] ;
                        cte[1] = xyr[1] + xyo[1] ;

                        xyo[0] += xyr[0] ;
                        xyo[1] += xyr[1] ;

                        n = 0 ;

                        break ;
                        }
                }
        else    {
                switch( cc.ch[1] ) {

                        case 72 :
                                xyr[1] -= np ;
                                if( xyr[1] < ( 1 - xymax[1] ) )
                                        xyr[1] = 1 - xymax[1] ;
                                if( ( xyo[1] + xyr[1] ) < 0 )
                                        xyr[1] = -xyo[1] ;
                                break ;
                        case 80 :
                                xyr[1] += np ;
                                if( xyr[1] > ( xymax[1] - 1 ) )
                                        xyr[1] = xymax[1] - 1 ;
                                if( ( xyo[1] + xyr[1] ) > ( pm[1] - xymax[1] ) ) {
                                        xyr[1] = pm[1] - xymax[1] - xyo[1] - 2 ;
                                        if( xyr[1] < 0 )
                                                xyr[1] = 0 ;
                                        }
                                break ;
                        case 77 :
                                xyr[0] += np ;
                                if( xyr[0] > ( xymax[0] - 1 ) )
                                        xyr[0] = xymax[0] - 1 ;
                                if( ( xyo[0] + xyr[0] ) > ( pm[0] - xymax[0] ) ) {
                                        xyr[0] = pm[0] - xymax[0] - xyo[0] - 2 ;
                                        if( xyr[0] < 0 )
                                                xyr[0] = 0 ;
                                        }
                                break ;
                        case 75 :
```

```
                                        xyr[0] -= np ;
                                        if( xyr[0] < ( 1 - xymax[0] ) )
                                                xyr[0] = 1 - xymax[0] ;
                                        if( ( xyo[0] + xyr[0] ) < 0 )
                                                xyr[0] = -xyo[0] ;
                                        break ;
                                }
                        break ;
                        }
                }
while( 1 ) ;

if( xya[0] <= 0 ) {

        if( xya[0] == 0 ) {

                if( xya[1] < 0 ) {

                        if( ( xya[1] + lp ) > 0 ) {
                                putimage(0,0,buff11,COPY_PUT) ;
                                free( buff11 ) ;
                                }

                        putimage(0,max(0,b),buff12,COPY_PUT) ;
                        free( buff12 ) ;
                        }
                else    {

                        putimage(0,xya[1],buff11,COPY_PUT) ;
                        free( buff11 ) ;

                        if( ( xya[1] - lp ) < 0 ) {
                                putimage(0,b,buff12,COPY_PUT) ;
                                free( buff12 ) ;
                                }
                        }
                }

        if( xya[1] < 0 ) {

                if( a >= 0 ) {
                        putimage(d,0,buff21,COPY_PUT) ;
                        free( buff21 ) ;
                        }

                putimage(d,max(0,b),buff22,COPY_PUT) ;
                free( buff22 ) ;
                }
        else    {

                putimage(d,xya[1],buff21,COPY_PUT) ;
                free( buff21 ) ;

                if( ( xya[1] - lp ) < 0 ) {
                        putimage(d,b,buff22,COPY_PUT) ;
                        free( buff22 ) ;
                        }
                }
        }

else    {
```

```
                if( xya[1] < 0 ) {

                        if( a >= 0 ) {
                                putimage(xya[0],0,buff11,COPY_PUT) ;
                                free( buff11 ) ;
                                }

                        putimage(xya[0],max(0,b),buff12,COPY_PUT) ;
                        free( buff12 ) ;
                        }
                else    {

                        putimage(xya[0],xya[1],buff11,COPY_PUT) ;
                        free( buff11 ) ;

                        if( ( xya[1] - lp ) < 0 ) {
                                putimage(xya[0],b,buff12,COPY_PUT) ;
                                free( buff12 ) ;
                                }
                        }
                }

        if( xya[1] <= 0 ) {

                if( xya[1] == 0 ) {
                        putimage(auxm2,0,buff3,COPY_PUT) ;
                        free( buff3 ) ;
                        }
                putimage(auxm2,c,buff4,COPY_PUT) ;
                free( buff4 ) ;
                }
        else    {
                putimage(auxm2,xya[1],buff3,COPY_PUT) ;
                free( buff3 ) ;
                }

        xya[0] = xyr[0] ;
        xya[1] = xyr[1] ;

        }
while( n ) ;

}

/*-------------------------------------------------------------------*/
/*      A61         DIBUJA SEMI-RECTANGULO DE CAMBIO DE CENTRO       */
/*-------------------------------------------------------------------*/

a61( lp, xyr, xymax )

int             lp      ;
int             xyr[ ]  ;
int             xymax[ ];

{

int             c       ;
int             d       ;
```

```
c = xyr[1] + xymax[1] ;
d = xyr[0] + xymax[0] ;

line( xyr[0], xyr[1], d, xyr[1] ) ;
line( xyr[0], c, d, c ) ;
line( xyr[0], xyr[1], xyr[0], ( xyr[1] + lp - 1 ) ) ;
line( xyr[0], ( xyr[1] + xymax[1] - lp + 1 ), xyr[0], c ) ;
line( d, xyr[1], d, ( xyr[1] + lp - 1 ) ) ;
line( d, ( xyr[1] + xymax[1] - lp + 1 ), d, c ) ;


}

/*-------------------------------------------------------------------*/
/*      A7                          CAMBIO DE PAGINA                 */
/*-------------------------------------------------------------------*/

a7( xymax, xyo, cte, pm, pag )

int             xymax[ ];
int             xyo[ ];
int             cte[ ]    ;
int             pm[ ]     ;
int             pag       ;

{

int             xyr[ ] = { 0, 0 } ;


if( ( cte[0] == 0 ) && ( cte[1] == 0 ) ) {
        xyo[0] = 0 ;
        xyo[1] = 0 ;
        }

switch( pag ) {
        case -3 :
                if( pm[0] > xymax[0] ) {
                        xyr[0] -= xymax[0] / 2 ;
                        if( ( xyo[0] + xyr[0] ) < 0 )
                                xyr[0] = -xyo[0] ;
                        }
                if( pm[1] > xymax[1] ) {
                        xyr[1] -= xymax[1] / 2 ;
                        if( ( xyo[1] + xyr[1] ) < 0 )
                                xyr[1] = -xyo[1] ;
                        }
                break ;
        case -2 :
                xyr[0] -= xymax[0] ;
                if( ( xyo[0] + xyr[0] ) < 0 )
                        xyr[0] = -xyo[0] ;
                break ;
        case -1 :
                xyr[1] -= xymax[1] ;
                if( ( xyo[1] + xyr[1] ) < 0 )
                        xyr[1] = -xyo[1] ;
                break ;
        case 1 :
```

```
                xyr[1] += xymax[1];
                if( ( xyo[1] + xyr[1] ) > ( pm[1] - xymax[1] ) ) {
                        xyr[1] = pm[1] - xymax[1] - xyo[1] - 2;
                        if( xyr[1] < 0 )
                                xyr[1] = 0;
                        }
                break;
        case 2:
                xyr[0] += xymax[0];
                if( ( xyo[0] + xyr[0] ) > ( pm[0] - xymax[0] ) ) {
                        xyr[0] = pm[0] - xymax[0] - xyo[0] - 2;
                        if( xyr[0] < 0 )
                                xyr[0] = 0;
                        }
                break;
        case 3:
                if( pm[1] > xymax[1] ) {
                        xyr[1] += xymax[1] / 2;
                        if( ( xyo[1] + xyr[1] ) > ( pm[1] - xymax[1] ) ) {
                                xyr[1] = pm[1] - xymax[1] - xyo[1] - 2;
                                if( xyr[1] < 0 )
                                        xyr[1] = 0;
                                }
                        }
                if( pm[0] > xymax[0] ) {
                        xyr[0] += xymax[0] / 2;
                        if( ( xyo[0] + xyr[0] ) > ( pm[0] - xymax[0] ) ) {
                                xyr[0] = pm[0] - xymax[0] - xyo[0] - 2;
                                if( xyr[0] < 0 )
                                        xyr[0] = 0;
                                }
                        }
        }

cte[0] = xyr[0] + xyo[0];
cte[1] = xyr[1] + xyo[1];

xyo[0] += xyr[0];
xyo[1] += xyr[1];

}
```

```
#include "head.h"
#include "dato.h"
#include <stdlib.h>

extern struct lectur        l       ;
extern struct corfil c      ;

/*-------------------------------------------------------------------*/
/*    A2          CREACION ESTRUCTURAS CORFIL Y DIBUJO          */
/*-------------------------------------------------------------------*/

a2( )

{

int             nhu0    ;


a21( &nhu0 ) ;

a22( nhu0 ) ;

}

/*-------------------------------------------------------------------*/
/*    A21                 CREACION ESTRUCTURA CORFIL                 */
/*-------------------------------------------------------------------*/

a21( nhu0 )

int             *nhu0   ;

{


c.id = l.id ;
c.name = l.name ;

a211( ) ;

a212( ) ;

a213( nhu0 ) ;

a214( ) ;

a215( ) ;

a216( ) ;

}

/*-------------------------------------------------------------------*/
/*    A211        ASIGNACION DE VALORES GENERALES DE CORFIL          */
/*-------------------------------------------------------------------*/

a211( )

{
```

```
struct inpout       *mem_inpout(int,char * );

c.ninp   = l.nentra ;
c.nout   = l.nsalid ;

c.ncajs  = l.ncajis ;
c.ncajc  = l.ncajic ;

c.nbloif = l.nblif ;
c.nbuc = l.nbuc ;
c.nrecur = l.nrecur ;
c.ninic = l.ninic ;

if( c.ninp > 0 )
        c.inp = mem_inpout( c.ninp, "a211(c.inp)" ) ;

if( c.nout > 0 )
        c.out = mem_inpout( c.nout, "a211(c.out)" ) ;

}


/*-------------------------------------------------------------------------*/
/*    A212        GENERACION DE CAJAS S Y C DE CORFIL                      */
/*-------------------------------------------------------------------------*/

a212( )

{

int                 i       ;
int                 n       ;
struct caja         *caj    ;
struct cajita   *cji  ;
struct caja         *mem_caja(int,char * );
struct inpout       *mem_inpout(int,char * );


if( c.ncajs > 0 )
        c.cs = mem_caja( c.ncajs, "a212(c.cs)" ) ;

if( c.ncajc > 0 )
        c.cc = mem_caja( c.ncajc, "a212(c.cc)" ) ;

for( i=0 ; i < c.ncajs ; i++ )          {

        caj = (c.cs+i) ;
        cji = (l.cs+i) ;

        caj->id = cji->id ;

        caj->pohor = cji->pohor ;
        caj->pover = cji->pover ;

        caj->ncadna = cji->ncadna ;

        caj->ninp   = cji->nentra ;
        caj->nout   = cji->nsalid ;

        caj->name = cji->name ;
```

```
        n = cji->nentra ;

        if( n > 0 )
                (c.cs+i)->inp = mem_inpout( n, "a212(c.cs->inp)" ) ;

        n = cji->nsalid ;

        if( n > 0 )
                (c.cs+i)->out = mem_inpout( n, "a212(c.cs->out)" ) ;
        }

for( i=0 ; i < c.ncajc ; i++ )          {

        caj = (c.cc+i) ;
        cji = (l.cc+i) ;

        caj->id = cji->id ;

        caj->pohor = cji->pohor ;
        caj->pover = cji->pover ;

        caj->ncadna = cji->ncadna ;

        caj->ninp   = cji->nentra ;
        caj->nout   = cji->nsalid ;

        caj->name = cji->name ;

        n = cji->nentra ;

        if( n > 0 )
                (c.cc+i)->inp = mem_inpout( n, "a212(c.cc->inp)" ) ;

        n = cji->nsalid ;

        if( n > 0 )
                (c.cc+i)->out = mem_inpout( n, "a212(c.cc->out)" ) ;

        }

}

/*-------------------------------------------------------------------*/
/*    A213        ORDEN DE I/O Y CREACION DE FLECHAS DATO             */
/*-------------------------------------------------------------------*/

a213( nhu0 )

int             *nhu0   ;

{

int             i       ;
int             j       ;
int             first   ;
int             cur     ;
int             nleft   ;
int             a2131( ) ;
int             a2132( ) ;
```

```
int             a2133( ) ;
struct caja     *caj     ;
struct cajita   *cji     ;


for( j=0 ; j < c.ninp ; j++ )
        (l.ent+j)->orden = 0 ;

cur = 0 ;
nleft = c.ninp ;

while( nleft > 0 )  {

        for( j=0 ; j < c.ninp ; j++ )
                if( (l.ent+j)->orden == 0 ) {
                        first = j ;
                        break ;
                        }

        for( j = first+1 ; j < c.ninp ; j++ )
                if( (l.ent+j)->orden == 0 )
                        if( a2132( "ext", l.ent+first, l.ent+j ) == 2 )
                                first = j ;

        (l.ent+first)->orden = 1 ;

        (c.inp+cur)->s = (l.ent+first)->nam ;

        cur++ ;
        nleft-- ;
        }

for( i=0 ; i < c.ncajs ; i++ )          {

        caj = (c.cs+i) ;
        cji = (l.cs+i) ;

        for( j=0 ; j < caj->nout ; j++ )
                (cji->sal+j)->orden = 0 ;

        cur = 0 ;
        nleft = caj->nout ;

        while( nleft > 0 )  {

                for( j=0 ; j < caj->nout ; j++ )
                        if( (cji->sal+j)->orden == 0 ) {
                                first = j ;
                                break ;
                                }

                for( j = first+1 ; j < caj->nout ; j++ )
                        if( (cji->sal+j)->orden == 0 )
                                if( a2132( cji->id, cji->sal+first, cji->sal+j ) == 2 )
                                        first = j ;

                (cji->sal+first)->orden = 1 ;

                (caj->out+cur)->s = (cji->sal+first)->nam ;

                cur++ ;
```

```
                nleft-- ;
                }
        }

for( i=0 ; i < c.ncajc ; i++ )          {

        caj = (c.cc+i) ;
        cji = (l.cc+i) ;

        for( j=0 ; j < caj->nout ; j++ )
                (cji->sal+j)->orden = 0 ;

        cur = 0 ;
        nleft = caj->nout ;

        while( nleft > 0 )  {

                for( j=0 ; j < caj->nout ; j++ )
                        if( (cji->sal+j)->orden == 0 ) {
                                first = j ;
                                break ;
                                }

                for( j = first+1 ; j < caj->nout ; j++ )
                        if( (cji->sal+j)->orden == 0 )
                                if( a2132( cji->id, cji->sal+first, cji->sal+j ) == 2 )
                                        first = j ;

                (cji->sal+first)->orden = 1 ;

                (caj->out+cur)->s = (cji->sal+first)->nam ;

                cur++ ;
                nleft-- ;
                }
        }

for( j=0 ; j < c.nout ; j++ )
        (c.out+j)->s = (l.sal+j)->nam ;

for( i=0 ; i < c.ncajs ; i++ )          {

        caj = (c.cs+i) ;
        cji = (l.cs+i) ;

        for( j=0 ; j < caj->ninp ; j++ )
                (cji->ent+j)->orden = 0 ;

        cur = 0 ;
        nleft = caj->ninp ;

        while( nleft > 0 )  {

                for( j=0 ; j < caj->ninp ; j++ )
                        if( (cji->ent+j)->orden == 0 ) {
                                first = j ;
                                break ;
                                }

                for( j = first+1 ; j < caj->ninp ; j++ )
                        if( (cji->ent+j)->orden == 0 )
```

```
                                        if( a2131( cji->ent+first, cji->ent+j ) == 2 )
                                                first = j ;

                        (cji->ent+first)->orden = 1 ;

                        (caj->inp+cur)->s = (cji->ent+first)->nam ;

                        cur++ ;
                        nleft-- ;
                        }
                }

for( i=0 ; i < c.ncajc ; i++ )          {

                caj = (c.cc+i) ;
                cji = (l.cc+i) ;

                for( j=0 ; j < caj->ninp ; j++ )
                        (cji->ent+j)->orden = 0 ;

                cur = 0 ;
                nleft = caj->ninp ;

                while( nleft > 0 )  {

                        for( j=0 ; j < caj->ninp ; j++ )
                                if( (cji->ent+j)->orden == 0 ) {
                                        first = j ;
                                        break ;
                                        }

                        for( j = first+1 ; j < caj->ninp ; j++ )
                                if( (cji->ent+j)->orden == 0 )
                                        if( a2131( cji->ent+first, cji->ent+j ) == 2 )
                                                first = j ;

                        (cji->ent+first)->orden = 1 ;

                        (caj->inp+cur)->s = (cji->ent+first)->nam ;

                        cur++ ;
                        nleft-- ;
                        }
                }

*nhu0 = a2133( ) ;

c.nfled = 0 ;

a2134( ) ;

}


/*-------------------------------------------------------------------------*/
/*     A2131        ORDEN RELATIVO DE DOS INPUTS                           */
/*-------------------------------------------------------------------------*/

int a2131( i1, i2 )

struct salida        *i1        ;
```

```
struct salida       *i2     ;

{

int                 i       ;
int                 f       ;
int                 k1      ;
int                 k2      ;
int                 no0( )  ;
int                 or0( )  ;
char                tip1    ;
char                tip2    ;
char                *ori1   ;
char                *ori2   ;
char                *nom1   ;
char                *nom2   ;


f = 0 ;

for( i=1 ; i < i1->norig ; i++ )
        if( or0( *(i1->tipo+f), *(i1->ori+f), *(i1->tipo+i), *(i1->ori+i) ) == 2 )
                f = i ;

tip1 = *(i1->tipo+f) ;
ori1 = *(i1->ori+f) ;
nom1 = *(i1->nom+f) ;

f = 0 ;

for( i=1 ; i < i2->norig ; i++ )
        if( or0( *(i2->tipo+f), *(i2->ori+f), *(i2->tipo+i), *(i2->ori+i) ) == 2 )
                f = i ;

tip2 = *(i2->tipo+f) ;
ori2 = *(i2->ori+f) ;
nom2 = *(i2->nom+f) ;

switch( or0( tip1, ori1, tip2, ori2 ) ) {

        case 0 :
                if( strcmp( ori1, "cod" ) == 0 ) {
                        if( strcmp( nom1, nom2 ) > 0 )
                                return( 1 ) ;
                        else
                                return( 2 ) ;
                        }
                k1 = no0( nom1, ori1, &i, &f ) ;
                k2 = no0( nom2, ori2, &i, &f ) ;
                if( k1 < k2 )
                        return( 1 ) ;
                else
                        return( 2 ) ;
        case 1 :
                return( 1 ) ;
        case 2 :
                return( 2 ) ;
        }
}
```

```
/*-------------------------------------------------------------------*/
/*    A2133        CALCULO DE HUECOS VERTICALES EN  SR=0                */
/*-------------------------------------------------------------------*/

int a2133( )

{

int             nhu0    ;
int             i       ;
int             j       ;


nhu0 = 0 ;

for( i=0 ; i < l.nentra ; i++ )
        for( j=0 ; j < (l.ent+i)->ndest ; j++ )
                if( *((l.ent+i)->tipo+j) == 'c' ) {
                        nhu0++ ;
                        break ;
                        }

return( nhu0 ) ;

}


/*-------------------------------------------------------------------*/
/*    A2134                CREACION DE FLECHAS DATO                    */
/*-------------------------------------------------------------------*/

a2134( )

{

int             i       ;
int             j       ;
int             k       ;
int             nrela   ;
char    aux[]={ "ext" } ;
struct cajita   *caj    ;
struct entrad   *ent    ;
struct relaci   *rela   ;
struct relaci   *mem_relaci(int,char * );
struct relaci   *memr_relaci(struct relaci * ,int,char * );


nrela = 0 ;
rela = mem_relaci( 1, "a2134(rela)" ) ;

for( j=0 ; j < l.nentra ; j++ ) {
        ent = (l.ent+j) ;
        for( k=0 ; k < ent->ndest ; k++ )
                if( *(ent->tipo+k) == 'i' ) {
                        rela = memr_relaci( rela, (nrela+1), "a2134(rela)" ) ;
                        (rela+nrela)->cajori = aux ;
                        (rela+nrela)->nomori = ent->nam ;
                        (rela+nrela)->cajdes = *(ent->des+k) ;
                        (rela+nrela)->nomdes = *(ent->nom+k) ;
                        (rela+nrela)->usada  = 0 ;
                        nrela++ ;
```

```
                        }
        }

for( i=0 ; i < l.ncajis ; i++ ) {
        caj = (l.cs+i) ;
        for( j=0 ; j < caj->nsalid ; j++ ) {
                ent = (caj->sal+j) ;
                for( k=0 ; k < ent->ndest ; k++ )
                        if( *(ent->tipo+k) == 'i' ) {
                                rela = memr_relaci( rela, (nrela+1), "a2134(rela)" ) ;
                                (rela+nrela)->cajori = caj->id ;
                                (rela+nrela)->nomori = ent->nam ;
                                (rela+nrela)->cajdes = *(ent->des+k) ;
                                (rela+nrela)->nomdes = *(ent->nom+k) ;
                                (rela+nrela)->usada  = 0 ;
                                nrela++ ;
                                }
                }
        }

for( i=0 ; i < l.ncajic ; i++ ) {
        caj = (l.cc+i) ;
        for( j=0 ; j < caj->nsalid ; j++ ) {
                ent = (caj->sal+j) ;
                for( k=0 ; k < ent->ndest ; k++ )
                        if( *(ent->tipo+k) == 'i' ) {
                                rela = memr_relaci( rela, (nrela+1), "a2134(rela)" ) ;
                                (rela+nrela)->cajori = caj->id ;
                                (rela+nrela)->nomori = ent->nam ;
                                (rela+nrela)->cajdes = *(ent->des+k) ;
                                (rela+nrela)->nomdes = *(ent->nom+k) ;
                                (rela+nrela)->usada  = 0 ;
                                nrela++ ;
                                }
                }
        }

a21341( nrela, rela ) ;

free( rela ) ;

}


/*-------------------------------------------------------------------*/
/*    A21341              CREACION DE FLECHAS DATO(2)                */
/*-------------------------------------------------------------------*/

a21341( nrela, rela )

int             nrela   ;
struct relaci   *rela   ;

{

int             i       ;
int             j       ;
int             k       ;
int             pohde   ;
int             pohdi   ;
int             ndes    ;
```

```
int             nori    ;
char            *chaux1 ;
char            *chaux2 ;
char            *(*des)  ;
char            *(*nomd);
char            *(*ori)  ;
char            *(*nomo);
struct relaci   *rel1    ;
struct relaci   *rel2    ;
int             ph0( )   ;
char            *(*mem_charp(int,char * ));
char            *(*memr_charp(char *(*) ,int,char * ));
struct fledat   *mem_fledat(int,char * );
struct fledat   *memr_fledat(struct fledat * ,int,char * );


for( i=0 ; i < nrela ; i++ ) {

        rel1 = (rela+i) ;
        if( rel1->usada == 0 ) {

                des  = mem_charp( 1, "a21341(des)" ) ;
                nomd = mem_charp( 1, "a21341(nomd)" ) ;
                ori  = mem_charp( 1, "a21341(ori)" ) ;
                nomo = mem_charp( 1, "a21341(nomo)" ) ;

                nori = 1 ;
                ndes = 0 ;
                *ori  = rel1->cajori ;
                *nomo = rel1->nomori ;

                for( j=i ; j < nrela ; j++ ) {

                        rel2 = (rela+j) ;
                        if( ( strcmp( rel2->nomori, rel1->nomori ) == NULL ) &&
                            ( strcmp( rel2->cajori, rel1->cajori ) == NULL ) &&
                            ( rel2->usada==0 ) ) {

                                if( strcmp( rel2->cajdes, "ext" ) == NULL ) {

                                        if( ndes == 0 ) {
                                                rel2->usada = 1 ;
                                                *des = rel2->cajdes ;
                                                *nomd = rel2->nomdes ;
                                                pohde = 0 ;
                                                ndes = 1 ;
                                                break ;
                                                }
                                        }

                                else    {

                                        rel2->usada = 1 ;
                                        des  = memr_charp( des, (ndes+1), "a21341(des)" ) ;
                                        nomd = memr_charp( nomd, (ndes+1), "a21341(nomd)" ) ;
                                        *(des+ndes)  = rel2->cajdes ;
                                        *(nomd+ndes) = rel2->nomdes ;

                                        if( ndes == 0 )
                                                pohde = ph0( rel2->cajdes ) ;
                                        else    {
```

```
                                        pohdi = ph0( rel2->cajdes ) ;
                                        pohde = ( pohdi < pohde ) ? pohdi : pohde ;
                                        }

                                    ndes++ ;
                                    }
                                }
                            }

                if( strcmp( rel1->cajori, "ext" ) != NULL )

                        for( j=0 ; j < ndes ; j++ ) {

                                chaux1 = *(des+j) ;
                                chaux2 = *(nomd+j) ;
                                for( k=i ; k < nrela ; k++ ) {

                                        rel2 = (rela+k) ;
                                        if( ( strcmp( rel2->cajdes, chaux1 ) == NULL ) &&
                                            ( strcmp( rel2->nomdes, chaux2 ) == NULL ) &&
                                            ( strcmp( rel2->cajori, "ext" ) != NULL ) &&
                                            ( (ph0( rel2->cajori ) < pohde) || (ndes == 1) ) &&
                                            ( rel2->usada == 0 ) ) {

                                                ori  = memr_charp( ori,  (nori+1), "a21341(ori)" ) ;
                                                nomo = memr_charp( nomo, (nori+1),
"a21341(nomo)" ) ;

                                                *(nomo+nori) = rel2->nomori ;
                                                *(ori+nori) = rel2->cajori ;
                                                rel2->usada = 1 ;
                                                nori++ ;
                                                }
                                        }
                                }

                if( c.nfled == 0 )
                        c.fd = mem_fledat( 1, "a21341(fd)" ) ;
                else
                        c.fd = memr_fledat( c.fd, (c.nfled+1), "a21341(fd)" ) ;

                a213411( nori, ori, nomo, ndes, des, nomd ) ;

                free( ori ) ;
                free( nomo ) ;
                free( des ) ;
                free( nomd ) ;

                c.nfled++ ;

                }
        }
}


/*-------------------------------------------------------------------*/
/*    A213411              FLECHA DATO EN CORFIL                     */
/*-------------------------------------------------------------------*/

a213411( nori, ori, nomo, ndes, des, nomd )
```

```
int             nori    ;
char            *(*ori) ;
char            *(*nomo);
int             ndes    ;
char            *(*des) ;
char            *(*nomd);

{

int             i       ;
int             kk      ;
char            tip     ;
int             ord     ;
int             nf      ;
int             no0( )  ;
int             no1( )  ;
int             ni0( )  ;
struct inpout   *(*mem_inpoutp(int,char * ));


nf = c.nfled ;

(c.fd+nf)->ndest = ndes ;
(c.fd+nf)->norig = nori ;                                   /*printf("\nNF = %d Ndest = %d Norig =
%d",nf,(c.fd+nf)->ndest,(c.fd+nf)->norig);printf("\nNdes = %d Nori = %d",ndes,nori) ;*/

/*for(i=0;i<nori;i++)printf("\nOri%d = %s Nom%d = %s",i,*(ori+i),i,*(nomo+i));*/

/*for(i=0;i<ndes;i++)printf("\nDes%d = %s Nom%d = %s",i,*(des+i),i,*(nomd+i));getch();*/

(c.fd+nf)->or = mem_inpoutp( nori, "a213403(c.fd->or)" ) ;
(c.fd+nf)->de = mem_inpoutp( ndes, "a213403(c.fd->de)" ) ;

for( i=0 ; i<nori ; i++ )   {

/*      kk = no1( *(nomo+i), *(ori+i), &tip, &ord ) ;

        if( tip == 'e' )
                (I.ent+kk)->usada = 1 ;
        else if( tip == 's' )
                ((I.cs+ord)->sal+kk)->usada = 1 ;
        else if( tip == 'c' )
                ((I.cc+ord)->sal+kk)->usada = 1 ;  */

        kk = no0( *(nomo+i), *(ori+i), &tip, &ord ) ;

        if( tip == 'e' )
                *((c.fd+nf)->or+i) = (c.inp+kk) ;
        else if( tip == 's' )
                *((c.fd+nf)->or+i) = ((c.cs+ord)->out+kk) ;
        else if( tip == 'c' )
                *((c.fd+nf)->or+i) = ((c.cc+ord)->out+kk) ;

        }

for( i=0 ; i<ndes ; i++ )   {

        kk = ni0( *(nomd+i), *(des+i), &tip, &ord ) ;

        if( tip == 'e' )
```

```
                *((c.fd+nf)->de+i) = (c.out+kk) ;
        else if( tip == 's' )
                *((c.fd+nf)->de+i) = ((c.cs+ord)->inp+kk) ;
        else if( tip == 'c' )
                *((c.fd+nf)->de+i) = ((c.cc+ord)->inp+kk) ;
        }

}

/*-------------------------------------------------------------------*/
/*    A214                  BLOQUES IF EN CORFIL                     */
/*-------------------------------------------------------------------*/

a214( )

{

int             n       ;
int             i       ;
int             j       ;
int             jj      ;
int             k       ;
char            tip     ;
int             ord     ;
struct bloqif   *bi     ;
struct blif     *bif    ;
struct macro    *mac    ;
int             a2141() ;
struct bloqif   *mem_bloqif(int,char * );
char            *mem_char(int,char * );
char            *(*mem_charp(int,char * ));
int             *mem_int(int,char * );
struct inpout   *mem_inpout(int,char * );
struct inpout   *(*mem_inpoutp(int,char * ));


if( c.nbloif > 0 )
        c.bi = mem_bloqif( c.nbloif, "a214(c.bi)" ) ;

for( i=0 ; i < c.nbloif ; i++ )         {

        bif = (l.bif+i);
        bi  = (c.bi+i) ;

        bi->hueh0 = a2141( i ) ;

        bi->hueh1  = bif->hueh1 ;
        bi->nivani = bif->nivani ;
        bi->nmac   = bif->nmac   ;
        bi->norcon = bif->oc.norig ;

        if( bi->norcon == 0 )
                bi->norcon = 1 ;

        n = bif->nmac ;

        (c.bi+i)->con    = mem_charp( n, "a214(c.bi->con)" ) ;
        (c.bi+i)->huev0  = mem_int( n, "a214(c.bi->huev0)" ) ;
        (c.bi+i)->huev1  = mem_int( n, "a214(c.bi->huev1)" ) ;
        (c.bi+i)->nivcon = mem_int( n, "a214(c.bi->nivcon)" ) ;
```

```
        (c.bi+i)->ncaj   = mem_int( n, "a214(c.bi->ncaj)" ) ;

        for( j=0 ; j < n ; j++ )     {

                mac = (bif->mac+j) ;

                *(bi->ncaj+j) = mac->ncaj ;
                *(bi->nivcon+j) = mac->nivcon ;

                a2142( mac, (bi->huev0+j), (bi->huev1+j) ) ;

                k = strlen( mac->con ) ;

                *((c.bi+i)->con+j) = mem_char( k+4, "a214(*c.bi->con)" ) ;

                *((*(bi->con+j))+0) = 'i';
                *((*(bi->con+j))+1) = 'f';
                *((*(bi->con+j))+2) = ' ';

                for( jj=0 ; jj<k ; jj++ )
                        *((*(bi->con+j))+jj+3) = *(mac->con+jj) ;
                }

        n = bi->norcon ;

        (c.bi+i)->oc = mem_inpoutp( n, "a214(c.bi->oc)" ) ;

        }

for( i=0 ; i < l.nblif ; i++ ){

        bif = (l.bif+i)      ;
        bi  = (c.bi+i)       ;

        n   = bi->norcon ;

        if( bif->oc.norig == 0 ) {

                *((c.bi+i)->oc) = mem_inpout( 1, "a214(*c.bi->oc)" ) ;

                (*((c.bi+i)->oc))->s = mem_char( 4, "a214(c.bi->oc->s)" ) ;

                *((*(bi->oc))->s+0) = 'c' ;
                *((*(bi->oc))->s+1) = 'o' ;
                *((*(bi->oc))->s+2) = 'd' ;
                }
        else
                for( j=0 ; j < n ; j++ )     {

                        k = no0( *(bif->oc.nom+j), *(bif->oc.ori+j), &tip, &ord ) ;

                        if( tip == 'e' )
                                *((bi->oc)+j) = (c.inp+k) ;
                        else if( tip == 's' )
                                *((bi->oc)+j) = ((c.cs+ord)->out+k) ;
                        else if( tip == 'c' )
                                *((bi->oc)+j) = ((c.cc+ord)->out+k) ;
                        }

        }
}
```

```
/*-------------------------------------------------------------------*/
/*    A2141              HUECO HORIZONTAL IZQUIERDO DE UN BLOQUE IF    */
/*-------------------------------------------------------------------*/

int       a2141( i )

int               i       ;

{

int               x       ;
int               n       ;
int               j       ;
struct macro      *mac    ;
int               ph0()   ;


mac = ((l.bif+i)->mac+0) ;

x = 30000 ;
for( j=0 ; j < mac->ncaj ; j++ ) {
          n = ph0( *(mac->caj+j) ) ;
          x = min( x, n ) ;
          }

return( x ) ;

}


/*-------------------------------------------------------------------*/
/*    A2142              HUECOS VERTICALES DE UNA MACRO                */
/*-------------------------------------------------------------------*/

a2142( mac, huev0, huev1 )

struct macro      *mac    ;
int               *huev0  ;
int               *huev1  ;

{

int               n       ;
int               k       ;
int               pv0()   ;


*huev0 = 30000 ;
*huev1 = 0 ;
for( k=0 ; k < mac->ncaj ; k++ ) {
          n = pv0( *(mac->caj+k) ) ;
          *huev0 = min( *huev0, n ) ;
          *huev1 = max( *huev1, n ) ;
          }

(*huev1)++ ;

}
```

```
/*-------------------------------------------------------------------*/
/*     A215                    BUCLES EN CORFIL                      */
/*-------------------------------------------------------------------*/

a215( )

{

int             n       ;
int             i       ;
int             j       ;
int             k       ;
int             ord     ;
char            tip     ;
char            *(*nam) ;
char            *(*ori) ;
struct bucli    *buc    ;
struct bucle    *bu     ;
int             ph0( )  ;
int             pv0( )  ;
char            *mem_char(int,char * );
struct bucle    *mem_bucle(int,char * );
struct inpout   *mem_inpout(int,char * );
struct inpout   *(*mem_inpoutp(int,char * ));


if( c.nbuc > 0 )
        c.bu  = mem_bucle( c.nbuc, "a215(c.bu)" ) ;

for( i=0 ; i < c.nbuc ; i++ )           {

        bu  = (c.bu+i)    ;
        buc = (l.buc+i)   ;

        a2151( i, &bu->hueh0, &bu->huev0, &bu->hueh1, &bu->huev1 ) ;

        bu->texaux = buc->texaux ;

        bu->texcon = buc->texcon ;

        bu->norcon = buc->oc.norig ;
        bu->norbu  = buc->ob.norig ;
        bu->nexbu  = buc->eb.norig ;

        if( bu->norcon == 0 )
                bu->norcon = 1 ;
        if( bu->norbu == 0 )
                bu->norbu = 1 ;
        if( bu->nexbu == 0 )
                bu->nexbu = 1 ;

        bu->tipo  = buc->tipo ;
        bu->nivcon  = buc->nivcon ;
        bu->nivani  = buc->nivani ;

        n = bu->norcon ;
        (c.bu+i)->oc = mem_inpoutp( n, "a215(c.bu->oc)" ) ;

        n = bu->norbu ;
        (c.bu+i)->ob = mem_inpoutp( n, "a215(c.bu.ob" ) ;
```

```
        n = bu->nexbu ;
        (c.bu+i)->eb = mem_inpoutp( n, "a215(c.bu->eb)" ) ;

        }

for( i=0 ; i < c.nbuc ; i++ )          {

        bu = (c.bu+i) ;
        buc = (l.buc+i) ;

        j = ph0( buc->excon ) ;
        k = pv0( buc->excon ) ;

        bu->extcon.ch.sr = 1 + 2 * j ;
        bu->extcon.cv.sr = 1 + 2 * k ;

        n = bu->norcon ;

        if( buc->oc.norig == 0 ) {

                *((c.bu+i)->oc) = mem_inpout( 1, "a215(*c.bu->oc)" ) ;

                (*((c.bu+i)->oc))->s = mem_char( 4, "a215(c.bu->oc->s)" ) ;

                *((*(bu->oc))->s+0) = 'c' ;
                *((*(bu->oc))->s+1) = 'o' ;
                *((*(bu->oc))->s+2) = 'd' ;
                }
        else
                for( j=0 ; j < n ; j++ )     {

                        k = no0( *(buc->oc.nom+j), *(buc->oc.ori+j), &tip, &ord ) ;

                        if( tip == 'e' )
                                *((bu->oc)+j) = (c.inp+k) ;
                        else if( tip == 's' )
                                *((bu->oc)+j) = ((c.cs+ord)->out+k) ;
                        else if( tip == 'c' )
                                *((bu->oc)+j) = ((c.cc+ord)->out+k) ;
                        }

        n = bu->norbu ;

        if( buc->ob.norig == 0 ) {

                *((c.bu+i)->ob) = mem_inpout( 1, "a215(*c.bu->ob)" ) ;

                (*((c.bu+i)->ob))->s = mem_char( 4, "a215(c.bu->ob->s)" ) ;

                *((*(bu->ob))->s+0) = 'c' ;
                *((*(bu->ob))->s+1) = 'o' ;
                *((*(bu->ob))->s+2) = 'd' ;
                }
        else
                for( j=0 ; j < n ; j++ ) {

                        k = no0( *(buc->ob.nom+j), *(buc->ob.ori+j), &tip, &ord ) ;

                        if( tip == 'e' )
                                *((bu->ob)+j) = (c.inp+k) ;
```

```
                        else if( tip == 's' )
                                *((bu->ob)+j) = ((c.cs+ord)->out+k) ;
                        else if( tip == 'c' )
                                *((bu->ob)+j) = ((c.cc+ord)->out+k) ;
                        }

        n = bu->nexbu ;

        if( buc->eb.norig == 0 ) {

                *((c.bu+i)->eb) = mem_inpout( 1, "a215(*c.bu->eb)" ) ;

                (*((c.bu+i)->eb))->s = mem_char( 4, "a215(c.bu->eb->s)" ) ;

                *((*(bu->eb))->s+0) = 'c' ;
                *((*(bu->eb))->s+1) = 'o' ;
                *((*(bu->eb))->s+2) = 'd' ;
                }
        else
                for( j=0 ; j < n ; j++ )    {

                        k = ni0( *(buc->eb.nom+j), *(buc->eb.ori+j), &tip, &ord ) ;

                        if( tip == 'e' )
                                *((bu->eb)+j) = (c.out+k) ;
                        else if( tip == 's' )
                                *((bu->eb)+j) = ((c.cs+ord)->inp+k) ;
                        else if( tip == 'c' )
                                *((bu->eb)+j) = ((c.cc+ord)->inp+k) ;
                        }
        }
}

/*-------------------------------------------------------------------*/
/*    A2151                        HUECOS DE UN BUCLE                */
/*-------------------------------------------------------------------*/

a2151( i, hueh0, huev0, hueh1, huev1 )

int             i       ;
int             *hueh0  ;
int             *huev0  ;
int             *hueh1  ;
int             *huev1  ;

{

int             nh      ;
int             nv      ;
int             k       ;
struct bucli    *buc    ;
int             pv0()   ;
int             ph0()   ;


buc = (l.buc+i) ;

*hueh0 = 30000 ;
*huev0 = 30000 ;
*hueh1 = 0 ;
```

```
*huev1 = 0 ;

for( k=0 ; k < buc->ncaj ; k++ ) {
        nh = ph0( *(buc->caj+k) ) ;
        nv = pv0( *(buc->caj+k) ) ;
        *hueh0 = min( *hueh0, nh ) ;
        *huev0 = min( *huev0, nv ) ;
        *hueh1 = max( *hueh1, nh ) ;
        *huev1 = max( *huev1, nv ) ;
        }

(*hueh1)++ ;
(*huev1)++ ;

}


/*-------------------------------------------------------------------------*/
/*    A216        RECURSIONES E INICIALIZACIONES EN CORFIL                 */
/*-------------------------------------------------------------------------*/

a216( )

{

int             i       ;
int             k       ;
int             ord     ;
char            tip     ;
struct cursi    *rec    ;
struct inici    *ini    ;
struct recurs   *re     ;
struct inicia   *in     ;
int             ni0( )  ;
int             ph0( )  ;
int             pv0( )  ;
struct recurs   *mem_recurs(int,char * );
struct inicia   *mem_inicia(int,char * );


if( c.nrecur > 0 )
        c.re  = mem_recurs( c.nrecur, "a216(c.re)" ) ;

if( c.ninic > 0 )
        c.in  = mem_inicia( c.ninic, "a216(c.in)" ) ;

for( i=0 ; i < c.nrecur ; i++ )        {

        rec = (l.rec+i) ;
        re = (c.re+i) ;

        re->texcon = rec->tex ;

        re->pohor = ph0( rec->caja ) ;
        re->pover = pv0( rec->caja ) ;

        }

for( i=0 ; i < c.ninic ; i++ )         {

        ini = (l.ini+i) ;
```

```
        in =  (c.in+i) ;

        in->valor = ini->val ;

        k = ni0( ini->inp, ini->caja, &tip, &ord ) ;

        if( tip == 's' )
                in->var = ((c.cs+ord)->inp+k) ;
        else if( tip == 'c' )
                in->var = ((c.cc+ord)->inp+k) ;

        in->hue = ph0( ini->anca ) ;

        }

}
```

```
#include "head.h"
#include "control.h"
#include <stdlib.h>
#include <graphics.h>

/*-------------------------------------------------------------------------*/
/*    MEO                    MENU QUE APARECE CON LA CARATULA              */
/*-------------------------------------------------------------------------*/

int       meO( un, xymax, col, imp, una, dum )

struct unidad      *un      ;
int                xymax[ ];
struct color       col      ;
struct impres      *imp     ;
struct unidad      *una     ;
char               dum[]    ;

{

static int i=0     ;
int                k        ;
int                x        ;
int                y        ;
int                xy[8]    ;
int                facy[2]  ;
int                pm[2]    ;
struct unidad      une      ;
unsigned size      ;
int                flag3    ;
void               *buf     ;
char      *texO[ ] = {
                   "Crear",
                   "Leer",
                   "Imprimir",
                   "Salir" } ;
char      *letO[ ] = {"C","L","I","S" } ;
int       posO[ ] = { 0, 0, 0, 0 } ;
void               *mem_buf(unsigned,char * );


k = 4 ;

x = un->tw * 13 ;
y = 5 * ( k + 1 ) * un->th / 2 ;

xy[0] = un->tw * 9 ;
xy[1] = xymax[1] - y - un->th * 8 ;
xy[2] = xy[0] + x ;
xy[3] = xy[1] ;
xy[4] = xy[2] ;
xy[5] = xy[1] + y ;
xy[6] = xy[0] ;
xy[7] = xy[5] ;

flag3 = 0 ;
size = imagesize( xy[0], xy[1], xy[4], xy[5] ) ;

do        {
          if( coreleft( ) > size + 2000 ) {
                    flag3 = 1 ;
```

```
                buf = mem_buf( size, "me0" ) ;
                getimage( xy[0], xy[1], xy[4], xy[5], buf ) ;
                }

        menu( un, col, tex0, let0, pos0, xy, k ) ;

        x = resp( un, xy, &i, k, 0, 0 ) ;

        if( flag3 == 1 ) {
                putimage( xy[0], xy[1], buf, COPY_PUT ) ;
                free( buf ) ;
                }
        else    {
                merell( xy, col ) ;
                }

        switch( x ) {
                case 1 :
                        if( me4( un, xymax, col, dum ) == -1 )
                                break ;
                case 0 :
                case 3 :
                        return( x ) ;
                case 2 :
                        impresion( un, xymax, col, imp, una, &une, facy, 0, pm ) ;
                }

        }
while( 1 ) ;

}

/*-------------------------------------------------------------------------*/
/*    ME1                  MENU PRINCIPAL                                  */
/*-------------------------------------------------------------------------*/

int     me1( flag2, un, xymax, col )

int             *flag2  ;
struct unidad   *un     ;
int             xymax[ ];
struct color    col     ;

{

static int i=0  ;
int             k       ;
int             x       ;
int             y       ;
int             xy[8]   ;
unsigned size   ;
int             flag3   ;
int             meg( )  ;
void            *buf    ;
char    *tex1[ ] = {
                "Crear",
                "Leer",
                "Modificar",
                "Escala",
                "Paleta",
```

```
                "Imprimir",
                "Guardar",
                "Salir" } ;
char    *let1[ ] = {"C","L","M","E","P","I","G","S" } ;
int     pos1[ ] = { 0, 0, 0, 0, 0, 0, 0, 0 } ;
void            *mem_buf(unsigned,char * );


k = 8 ;
mecoor( un, xymax, xy, k, 9, 0 ) ;

flag3 = 0 ;
size = imagesize( xy[0], xy[1], xy[4], xy[5] ) ;

if( coreleft( ) > size + 2000 ) {
        flag3 = 1 ;
        buf = mem_buf( size, "me1" ) ;
        getimage( xy[0], xy[1], xy[4], xy[5], buf ) ;
        }

menu( un, col, tex1, let1, pos1, xy, k ) ;

do      {
        x = resp( un, xy, &i, k, 0, 1 ) ;

        if( x != 6 )
                break ;

        meg( un, xymax, col ) ;
        *flag2 = 0 ;

        }
while( 1 ) ;

if( flag3 == 1 ) {
        putimage( xy[0], xy[1], buf, COPY_PUT ) ;
        free( buf ) ;
        }
else    {
        merell( xy, col ) ;
        }
return( x ) ;

}


/*-------------------------------------------------------------------*/
/*      MEG                         AVISO DE "GUARDANDO"             */
/*-------------------------------------------------------------------*/

int     meg( un, xymax, col )

struct unidad   *un     ;
int             xymax[ ];
struct color    col     ;

{

int             xy[8]   ;
unsigned size   ;
int             flag3   ;
```

```
void    .           *buf     ;
char     *tex11[ ]  = { "< GUARDANDO >" } ;
char     *let11[ ]  = {" "} ;
int      pos11[ ]   = { 0 } ;
void                *mem_buf(unsigned,char * );


mecoor( un, xymax, xy, 1, 13, 3 ) ;

flag3 = 0 ;
size = imagesize( xy[0], xy[1], xy[4], xy[5] ) ;

if( coreleft( ) > size ) {
          flag3 = 1 ;
          buf = mem_buf( size, "me11" ) ;
          getimage( xy[0], xy[1], xy[4], xy[5], buf ) ;
          }

menu( un, col, tex11, let11, pos11, xy, 1 ) ;

a1( ) ;

if( flag3 = = 1 ) {
          putimage( xy[0], xy[1], buf, COPY_PUT ) ;
          free( buf ) ;
          }
else      {
          merell( xy, col ) ;
          }

}
```

```
#include "head.h"
#include <stdlib.h>
#include <stdio.h>
#include <dir.h>
#include <dos.h>
#include <graphics.h>

extern struct lectur l ;

/*-------------------------------------------------------------------*/
/*      ME5                          MENU DE MODIFICAR                */
/*-------------------------------------------------------------------*/

int     me5( un, xymax, col )

struct unidad     *un     ;
int               xymax[ ];
struct color      col     ;

{

static int i=0    ;
static int volver=0;
int               k       ;
int               x       ;
int               y       ;
int               xy[8]   ;
int               flag3   ;
unsigned size     ;
int               me51( ) ;
int               me52( ) ;
int               me53( ) ;
int               me54( ) ;
FILE              *fp     ;
char    *tex5[ ] = {
                  "Poner",
                  "Quitar",
                  "Cambiar",
                  "ARBOL" } ;
char    *let5[ ] = {"P","Q","C","A"} ;
int     pos5[ ] = { 0, 0, 0, 0 } ;


k = 4 ;
mecoor( un, xymax, xy, k, 7, 1 ) ;

flag3 = 0 ;
size = imagesize( xy[0], xy[1], xy[4], xy[5] ) ;

if( coreleft( ) > ( size + 4000 ) ) {
        flag3 = 1 ;
        if( ( fp = tmpfile( ) ) == NULL )
                { closegraph( ) ; perror("error: me5: tmpfile: ") ; exit( 1 ) ; }
        mecoj( fp, xy, size ) ;
        }

menu( un, col, tex5, let5, pos5, xy, k ) ;

do      {
        y = 1 ;
        if( volver ) {
```

```
                x = volver - 1 ;
                volver = 0 ;
                }
        else
                x = resp( un, xy, &i, k, 0, 5 ) ;

        switch( x ) {
                case -2 :
                case -1 :
                        y = 0 ;
                        break ;
                case 0 :
                        if( ( x = me51( un, xymax, col ) ) == -2 ) {
                                y = 0 ;
                                volver = 1 ;
                                }
                        break ;
                case 1 :
                        if( ( x = me52( un, xymax, col ) ) == -2 ) {
                                y = 0 ;
                                volver = 2 ;
                                }
                        break ;
                case 2 :
                        if( ( x = me53( un, xymax, col ) ) == -2 ) {
                                y = 0 ;
                                volver = 3 ;
                                }
                        break ;
                case 3 :
                        if( ( x = me54( un, xymax, col ) ) == -2 ) {
                                y = 0 ;
                                volver = 4 ;
                                }
                }
        }
while( y ) ;

if( flag3 == 1 ) {
        mepon( fp, xy, size ) ;
        fclose( fp ) ;
        }
else    {
        merell( xy, col ) ;
        }
return( x ) ;

}


/*-------------------------------------------------------------------------*/
/*      ME54                            MENU ARBOL                         */
/*-------------------------------------------------------------------------*/

int     me54( un, xymax, col )

struct unidad   *un     ;
int             xymax[ ];
struct color    col     ;

{
```

```
static int i=0      ;
int                 k       ;
int                 x       ;
int                 y       ;
int                 xy[8]   ;
int                 flag3   ;
unsigned size       ;
int                 me541( );
int                 me542( );
FILE                *fp     ;
char    *tex54[ ] = {
                    "Ident.",
                    "Nombre",
                    "Entrada",
                    "Salida" } ;
char    *let54[ ] = {"I","N","E","S"} ;
int     pos54[ ] = { 0, 0, 0, 0 } ;


k = 4 ;
mecoor( un, xymax, xy, k, 7, 2 ) ;

flag3 = 0 ;
size = imagesize( xy[0], xy[1], xy[4], xy[5] ) ;

if( coreleft( ) > ( size + 3000 ) ) {
        flag3 = 1 ;
        if( ( fp = tmpfile( ) ) == NULL )
                { closegraph( ) ; perror("error: me54: tmpfile: ") ; exit( 1 ) ; }
        mecoj( fp, xy, size ) ;
        }

menu( un, col, tex54, let54, pos54, xy, k ) ;

do      {
        y = 1 ;
        x = resp( un, xy, &i, k, 0, 54 ) ;

        switch( x ) {
                case -2 :
                case -1 :
                        y = 0 ;
                        break ;
                case 0 :
                        me541( un, xymax, col ) ;
                        break ;
                case 1 :
                        me542( un, xymax, col ) ;
                        break ;
                case 2 :
                        medato( un, xymax, col, "ext", 'e', -1, 0 ) ;
                        break ;
                case 3 :
                        medato( un, xymax, col, "ext", 'e', -1, 1 ) ;
                }
        }
while( y ) ;

if( flag3 == 1 ) {
        mepon( fp, xy, size ) ;
```

```
        fclose( fp ) ;
        }
else    {
        merell( xy, col ) ;
        }

return( x ) ;

}


/*-------------------------------------------------------------------*/
/*      ME541           VENTANA DE IDENTIFICADOR                     */
/*-------------------------------------------------------------------*/

int     me541( un, xymax, col )

struct unidad   *un     ;
int             xymax[ ];
struct color    col     ;

{

int             k       ;
int             x       ;
int             y       ;
int             xy[8]   ;
int             curxy[2];
int             flag    ;
int             flag3   ;
unsigned size   ;
char            dum[9]  ;
char            dub[13] ;
char            drive[3];
char            dir[3]  ;
char            file[9] ;
char            ext[5]  ;
int             lecad( );
FILE            *fp     ;
void            *buf    ;
struct ffblk    ffblk   ;
char    *tex541[ ] = {
                " Identificador >",
                "     ¡No es valido!" } ;
char    *let541[ ] = {" "," "} ;
int     pos541[ ] = { 0, 0 } ;
char            *memr_char(char * ,int,char * );
void            *mem_buf(unsigned,char * );


k = 1 ;

do      {
        if( k == -1 )
                k = 1 ;

        mecoor( un, xymax, xy, k, 26, 3 ) ;

        flag3 = 0 ;
        size = imagesize( xy[0], xy[1], xy[4], xy[5] ) ;
```

```
        if( coreleft( ) > ( size + 1000 ) ) {
                flag3 = 1 ;
                buf = mem_buf( size, "me541" ) ;
                getimage( xy[0], xy[1], xy[4], xy[5], buf ) ;
                }

        menu( un, col, tex541, let541, pos541, xy, k ) ;

        curxy[0] = xy[0] + un->tw * 20 ;
        curxy[1] = xy[1] + un->th * 3 ;

        do      {
                if( lecad( un, col, curxy, dum, 8 ) != -1 )
                        if( ( y = strlen( dum ) ) > 0 ) {
                                flag = fnsplit( dum, drive, dir, file, ext ) ;
                                if( (flag&DRIVE) || (flag&DIRECTORY) || (flag&FILENAME) ||
(flag&EXTENSION) )
                                        strcpy( dum, file ) ;
                                strcpy( dub, dum ) ;
                                if( verid( un, xymax, col, dub ) == -1 ) {
                                        k = -1 ;
                                        break ;
                                        }
                                strcat( dub, ".ana" ) ;
                                if( !findfirst( dub, &ffblk, FA_ARCH ) ) {
                                        if( me5411( un, xymax, col ) == 1 ) {
                                                k = -1 ;
                                                break ;
                                                }
                                        }
                                else    {
                                        if( flag & FILENAME ) {
                                                if( ( fp = fopen( dub, "w" ) ) == NULL ) {
                                                        k = 2 ;
                                                        break ;
                                                        }
                                                fclose( fp ) ;
                                                remove( dub ) ;
                                                }
                                        else    {
                                                k = 2 ;
                                                break ;
                                                }
                                        }
                                l.id = memr_char( l.id, (y+1), "me541(l.id)" ) ;
                                strcpy( l.id, dum ) ;
                                k = 1 ;
                                break ;
                                }
                k = 1 ;
                break ;
                }
        while( 1 ) ;

        if( flag3 == 1 ) {
                putimage( xy[0], xy[1], buf, COPY_PUT ) ;
                free( buf ) ;
                }
        else    {
                merell( xy, col ) ;
                }
```

```
        if( k == 1 )
                break ;

        }
while( 1 ) ;

}

/*-------------------------------------------------------------------*/
/*      ME542                       VENTANA DE NOMBRE                */
/*-------------------------------------------------------------------*/

int     me542( un, xymax, col )

struct unidad   *un     ;
int             xymax[ ];
struct color    col     ;

{

int             k       ;
int             x       ;
int             y       ;
int             xy[8]   ;
int             curxy[2];
int             flag3   ;
unsigned size   ;
char            dum[41] ;
int             lecad( );
void            *buf    ;
char    *tex542[ ] = {
                " Nombre  >" } ;
char    *let542[ ] = {" "} ;
int     pos542[ ] = { 0 } ;
char            *memr_char(char * ,int,char * );
void            *mem_buf(unsigned,char * );


k = 1 ;
mecoor( un, xymax, xy, k, 51, 3 ) ;

flag3 = 0 ;
size = imagesize( xy[0], xy[1], xy[4], xy[5] ) ;

if( coreleft( ) > ( size + 1000 ) ) {
        flag3 = 1 ;
        buf = mem_buf( size, "me542" ) ;
        getimage( xy[0], xy[1], xy[4], xy[5], buf ) ;
        }

menu( un, col, tex542, let542, pos542, xy, k ) ;

curxy[0] = xy[0] + un->tw * 13 ;
curxy[1] = xy[1] + un->th * 3 ;

if( lecad( un, col, curxy, dum, 40 ) != -1 )
        if( ( y = strlen( dum ) ) > 0 ) {
                l.name = memr_char( l.name, (y+1), "me542(l.name)" ) ;
                strcpy( l.name, dum ) ;
```

```
                    }
if( flag3 == 1 ) {
          putimage( xy[0], xy[1], buf, COPY_PUT ) ;
          free( buf ) ;
          }
else      {
          merell( xy, col ) ;
          }

}

/*-------------------------------------------------------------------*/
/*    ME5411                MENU DE EXISTIR                          */
/*-------------------------------------------------------------------*/

int       me5411( un, xymax, col )

struct unidad     *un      ;
int               xymax[ ];
struct color      col      ;

{

int               x        ;
int               y        ;
int               xy[8]    ;
int               flag3    ;
unsigned size     ;
void              *buf     ;
union inkey       {
                  char     ch[2]    ;
                  int      i        ;
                  }        c        ;
void              *mem_buf(unsigned,char * );


x = un->tw * 32 ;
y = un->th * 7 ;

xy[0] = ( xymax[0] + 1 ) / 2 - x / 2 ;
xy[1] = xymax[1] / 2 - y ;
xy[2] = xy[0] + x ;
xy[3] = xy[1] ;
xy[4] = xy[2] ;
xy[5] = xy[1] + y ;
xy[6] = xy[0] ;
xy[7] = xy[5] ;

flag3 = 0 ;
size = imagesize( xy[0], xy[1], xy[4], xy[5] ) ;
if( coreleft( ) > ( size + 1000 ) ) {
          flag3 = 1 ;
          buf = mem_buf( size, "me5411" ) ;
          getimage( xy[0], xy[1], xy[4], xy[5], buf ) ;
          }

setcolor( col.reve ) ;
setfillstyle( SOLID_FILL, col.fove ) ;
setlinestyle( SOLID_LINE, 0, NORM_WIDTH ) ;
```

```
fillpoly( 4, xy ) ;
setlinestyle( SOLID_LINE, 0, THICK_WIDTH ) ;
rectangle((xy[0]+un->tw/2),(xy[1]+un->th/2),(xy[4]-un->tw/2),(xy[5]-un->th/2)) ;
setcolor( col.teve ) ;
settextjustify( LEFT_TEXT, BOTTOM_TEXT ) ;
settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 ) ;

outtextxy( (xy[0]+(2*un->tw)), (xy[1]+(3*un->th)), " < Este fichero ya existe > " ) ;
outtextxy( (xy[0]+(2*un->tw)), (xy[1]+(11*un->th)/2), " ¿Desea machacarlo? (S/N)" ) ;

do      {
        while( !bioskey( 1 ) ) ;
        c.i = bioskey( 0 ) ;

        if( c.ch[0] ) {
                if( c.ch[0] == '\r' ) {
                        x = 1 ;
                        break ;
                        }
                else if( ( c.ch[0] == 's' ) || ( c.ch[0] == 'S' ) ) {
                        x = 0 ;
                        break ;
                        }
                else if( ( c.ch[0] == 'n' ) || ( c.ch[0] == 'N' ) ) {
                        x = 1 ;
                        break ;
                        }
                }
        }
while( 1 ) ;

if( flag3 == 1 ) {
        putimage( xy[0], xy[1], buf, COPY_PUT ) ;
        free( buf ) ;
        }
else    {
        merell( xy, col ) ;
        }

return( x ) ;

}


/*-------------------------------------------------------------------*/
/*    MEDATO                        MENU DE DATOS                    */
/*-------------------------------------------------------------------*/

int     medato( un, xymax, col, noc, tipcaj, orden, ensa )

struct unidad   *un             ;
int             xymax[ ] ;
struct color    col             ;
char            noc[ ]          ;
char            tipcaj          ;
int             orden           ;
int             ensa            ;

{

int             k               ;
```

```
int             i               ;
int             x               ;
int             y               ;
int             opc             ;
int             nele            ;
int             xy[8]           ;
int             flag3           ;
unsigned size           ;
FILE            *fp             ;
char            nod[2][21]      ;
struct cajita   *cji            ;
char            *(*texi)  ;
char            (*text)[11]     ;
char    titulo[ ] = {"DATO"}    ;
union inkey     {
                char    ch[2]   ;
                int     i       ;
                }       c       ;
char            *(*mem_charp(int,char * ));
char            (*mem_charvec(int,char * ))[11];


if( ensa = = 0 ) {
        if( tipcaj = = 'e' ) {
                if( l.nentra > 0 ) {
                        texi = mem_charp( (l.nentra), "medato(texi-e)" ) ;
                        for( i=0 ; i < l.nentra ; i+ + )
                                *(texi+i) = (l.ent+i)->nam ;
                        }
                else
                        return ;
                }
        else    {
                if( tipcaj = = 's' )
                        cji = (l.cs+orden) ;
                else
                        cji = (l.cc+orden) ;
                if( cji->nsalid > 0 ) {
                        texi = mem_charp( (cji->nsalid), "medato(texi-s)" ) ;
                        for( i=0 ; i < cji->nsalid ; i+ + )
                                *(texi+i) = (cji->sal+i)->nam ;
                        }
                else
                        return ;
                }
        }
else    {
        if( tipcaj = = 'e' ) {
                if( l.nsalid > 0 ) {
                        texi = mem_charp( (l.nsalid), "medato(texi-s)" ) ;
                        for( i=0 ; i < l.nsalid ; i+ + )
                                *(texi+i) = (l.sal+i)->nam ;
                        }
                else
                        return ;
                }
        else    {
                if( tipcaj = = 's' )
                        cji = (l.cs+orden) ;
                else
                        cji = (l.cc+orden) ;
```

```
                if( cji->nentra > 0 ) {
                        texi = mem_charp( (cji->nentra), "medato(texi-e)" ) ;
                        for( i=0 ; i < cji->nentra ; i++ )
                                *(texi+i) = (cji->ent+i)->nam ;
                        }
                else
                        return ;
                }
        }

if( i == 1 ) {
        strcpy( nod[0], *(texi+0) ) ;
        free( texi ) ;
        if( menoda( un, xymax, col, nod[1], 1 ) != -1 )
                canoda( noc, nod, ensa ) ;
        return ;
        }

nele = i ;
text = mem_charvec( nele, "medato(text)" ) ;

for( i=0 ; i < nele ; i++ )
        strncpy( *(text+i), *(texi+i), 10 ) ;

if( nele < 9 ) {
        opc = 0 ;
        k = nele + 1 ;
        mecoor( un, xymax, xy, k, 10, 4 ) ;
        k-- ;
        }
else    {
        opc = 1 ;
        mecoor( un, xymax, xy, 9, 10, 4 ) ;
        k = 0 ;
        }

flag3 = 0 ;
size = imagesize( xy[0], xy[1], xy[4], xy[5] ) ;

if( coreleft( ) > ( size + 2000 ) ) {
        flag3 = 1 ;
        if( ( fp = tmpfile( ) ) == NULL )
                { closegraph( ) ; perror("error: me543: tmpfile: ") ; exit( 1 ) ; }
        mecoj( fp, xy, size ) ;
        }

i=0 ;

do      {
        y = 0 ;
        mevar( un, col, text, titulo, xy, k, opc ) ;

        x = revar( un, xy, &i, k, opc ) ;

        switch( x ) {
                case -2 :
                        y=1 ;
                        i=0 ;
                        k -= 8 ;
                        if( k <= 0 ) {
                                k=0 ;
```

```
                                opc = 1 ;
                                }
                        else
                                opc = 2 ;
                        break ;
                case -3 :
                        y=1 ;
                        i=0 ;
                        k += 8 ;
                        if( k > nele - 8 ) {
                                k = nele - 8 ;
                                opc = 3 ;
                                }
                        else
                                opc = 2 ;
                case -1 :
                        break ;
                default :
                        if( opc == 0 )
                                strcpy( nod[0], *(texi+x) ) ;
                        else
                                strcpy( nod[0], *(texi+k+x) ) ;
                        if( menoda( un, xymax, col, nod[1], 1 ) == -1 )
                                y=1 ;
                        else
                                canoda( noc, nod, ensa ) ;
                }
        }
while( y ) ;

if( flag3 == 1 ) {
        mepon( fp, xy, size ) ;
        fclose(fp);
        }
else    {
        merell( xy, col ) ;
        }

free( text ) ;
free( texi ) ;

}

/*-----------------------------------------------------------------*/
/*    CANODA                       CAMBIO DE NOMBRE DE DATO        */
/*-----------------------------------------------------------------*/

canoda( noc, nod, ensa )

char            noc[]           ;
char            nod[][21];
int             ensa            ;

{

int             i'              ;
int             pos             ;
int             orden           ;
char            tipcaj          ;
struct entrad   *ent            ;
```

```
struct salida       *sal                ;
char                *memr_char(char * ,int,char * );


if( ensa == 0 ) {

        pos = no1( nod[0], noc, &tipcaj, &orden ) ;
        if( tipcaj == 'e' ) {
                ent = (l.ent+pos) ;
                (l.ent+pos)->nam = memr_char( (l.ent+pos)->nam, (strlen(nod[1]) + 1), "canoda(l.ent-
>nam)" ) ;
                }
        else if( tipcaj == 's' ) {
                ent = ((l.cs+orden)->sal+pos) ;
                ((l.cs+orden)->sal+pos)->nam = memr_char( ((l.cs+orden)->sal+pos)->nam,
(strlen(nod[1]) + 1), "canoda(l.cs->sal->nam)" ) ;
                }
        else if( tipcaj == 'c' ) {
                ent = ((l.cc+orden)->sal+pos) ;
                ((l.cc+orden)->sal+pos)->nam = memr_char( ((l.cc+orden)->sal+pos)->nam,
(strlen(nod[1]) + 1), "canoda(l.cc->sal->nam)" ) ;
                }

        strcpy( ent->nam, nod[1] ) ;
        for( i=0 ; i < ent->ndest ; i++ )
                canoda1( noc, nod, *(ent->des+i), *(ent->nom+i), *(ent->tipo+i) ) ;
        }
else    {

        pos = ni1( nod[0], noc, &tipcaj, &orden ) ;
        if( tipcaj == 'e' ) {
                sal = (l.sal+pos) ;
                (l.sal+pos)->nam = memr_char( (l.sal+pos)->nam, (strlen(nod[1]) + 1), "canoda(l.sal-
>nam)" ) ;
                }
        else if( tipcaj == 's' ) {
                sal = ((l.cs+orden)->ent+pos) ;
                ((l.cs+orden)->ent+pos)->nam = memr_char( ((l.cs+orden)->ent+pos)->nam,
(strlen(nod[1]) + 1), "canoda(l.cs->ent->nam)" ) ;
                }
        else if( tipcaj == 'c' ) {
                sal = ((l.cc+orden)->ent+pos) ;
                ((l.cc+orden)->ent+pos)->nam = memr_char( ((l.cc+orden)->ent+pos)->nam,
(strlen(nod[1]) + 1), "canoda(l.cc->ent->nam)" ) ;
                }

        strcpy( sal->nam, nod[1] ) ;
        for( i=0 ; i < sal->norig ; i++ )
                canoda2( noc, nod, *(sal->ori+i), *(sal->nom+i), *(sal->tipo+i) ) ;
        }

}


/*-------------------------------------------------------------------*/
/*    CANODA1          CAMBIO DEL DATO          EN DESTINOS          */
/*-------------------------------------------------------------------*/

canoda1( noc, nod, des, nam, tipo )

char                noc[]               ;
```

```
char            nod[][21];
char            *des            ;
char            *nam            ;
char            tipo            ;

{

int             i               ;
int             j               ;
int             pos             ;
int             orden           ;
char            tipcaj          ;
struct entrad   *ent            ;
struct salida   *sal            ;
struct condic   *co             ;
char            *memr_char(char * ,int,char * );
                                                        /*printf("\nNoc = %s
Nod[0] = %s Nod[1] = %s",noc,nod[0],nod[1]);*/
                                                        /*printf("\nDes = %s
Nom = %s Tipo = %c",des,nam,tipo);getch();*/
if( tipo == 'c' ) {
        j = 0 ;
        for( i=0 ; i < l.nbuc ; i++ )
                if( strcmp( des, (l.buc+i)->excon ) == NULL ) {
                        co = &((l.buc+i)->oc) ;
                        for( j=0 ; j < co->norig ; j++ )
                                if( (strcmp( *(co->nom+j), nod[0] ) == NULL) && (strcmp( *(co-
>ori+j), noc ) == NULL) ) {
                                        *((l.buc+i)->oc.nom+j) = memr_char( *((l.buc+i)-
>oc.nom+j), (strlen(nod[1]) +1), "canoda1(*l.buc->oc.nom)" ) ;
                                        strcpy( *(co->nom+j), nod[1] ) ;
                                        j = -1 ;
                                        break ;
                                        }
                        if( j == -1 )
                                break ;
                        }
        if( j != -1 )
                for( i=0 ; i < l.nblif ; i++ ) {
                        co = &((l.bif+i)->oc) ;
                        for( j=0 ; j < co->norig ; j++ )
                                if( (strcmp( *(co->nom+j), nod[0] ) == NULL) && (strcmp( *(co-
>ori+j), noc ) == NULL) ) {
                                        *((l.bif+i)->oc.nom+j) = memr_char( *((l.bif+i)-
>oc.nom+j), (strlen(nod[1]) +1), "canoda1(*l.bif->oc.nom)" ) ;
                                        strcpy( *(co->nom+j), nod[1] ) ;
                                        j = -1 ;
                                        break ;
                                        }
                        if( j == -1 )
                                break ;
                        }
        }
else    {
        pos = ni1( nam, des, &tipcaj, &orden ) ;

        if( pos != -1 ) {
                if( tipcaj == 'e' ) {
                        sal = (l.sal+pos) ;
                        for( i=0 ; i < sal->norig ; i++ )
```

```
                                        if( ( strcmp( *(sal->nom+i), nod[0] ) == NULL ) && ( strcmp( *(sal-
>ori+i), noc ) == NULL ) ) {
                                                *((l.sal+pos)->nom+i) = memr_char( *((l.sal+pos)-
>nom+i), (strlen(nod[1]) +1), "canoda1(*l.sal->nom)" ) ;
                                                strcpy( *(sal->nom+i), nod[1] ) ;
                                                break ;
                                                }
                                }

                        else if( tipcaj == 's' ) {
                                sal = ((l.cs+orden)->ent+pos) ;
                                for( j=0 ; j < sal->norig ; j++ )
                                        if( ( strcmp( *(sal->nom+j), nod[0] ) == NULL ) && ( strcmp( *(sal-
>ori+j), noc ) == NULL ) ) {
                                                *(((l.cs+orden)->ent+pos)->nom+j) = memr_char(
*(((l.cs+orden)->ent+pos)->nom+j), (strlen(nod[1]) +1), "canoda1(*l.cs->ent->nom)" ) ;
                                                strcpy( *(sal->nom+j), nod[1] ) ;
                                                break ;
                                                }
                                }

                        else if( tipcaj == 'c' ) {
                                sal = ((l.cc+orden)->ent+pos) ;
                                for( j=0 ; j < sal->norig ; j++ )
                                        if( ( strcmp( *(sal->nom+j), nod[0] ) == NULL ) && ( strcmp( *(sal-
>ori+j), noc ) == NULL ) ) {
                                                *(((l.cc+orden)->ent+pos)->nom+j) = memr_char(
*(((l.cc+orden)->ent+pos)->nom+j), (strlen(nod[1]) +1), "canoda1(l.cc->ent->nom)" ) ;
                                                strcpy( *(sal->nom+j), nod[1] ) ;
                                                break ;
                                                }
                                }
                        }

                if( tipo == 'b' )
                        for( i=0 ; i < l.nbuc ; i++ ) {
                                co = &((l.buc+i)->ob) ;
                                for( j=0 ; j < co->norig ; j++ )
                                        if( (strcmp( *(co->nom+j), nod[0] ) == NULL) && (strcmp( *(co-
>ori+j), noc ) == NULL) ) {
                                                *((l.buc+i)->ob.nom+j) = memr_char( *((l.buc+i)-
>ob.nom+j), (strlen(nod[1]) +1), "canoda1(*l.buc->ob.nom)" ) ;
                                                strcpy( *(co->nom+j), nod[1] ) ;
                                                i = l.nbuc ;
                                                break ;
                                                }
                                }
                }
}

/*-------------------------------------------------------------------*/
/*    CANODA2          CAMBIO DEL DATO          EN ORIGENES          */
/*-------------------------------------------------------------------*/

canoda2( noc, nod, ori, nam, tipo )

char            noc[]           ;
char            nod[][21];
char            *ori            ;
char            *nam            ;
```

```
char            tipo            ;

{

int             i               ;
int             j               ;
int             pos             ;
int             orden           ;
char            tipcaj          ;
struct entrad   *ent            ;
struct salida   *sal            ;
struct inici    *ini            ;
struct condic   *co             ;
char            *memr_char(char * ,int,char * );
                                                        /*printf("\nNoc = %s
Nod[0] = %s Nod[1] = %s",noc,nod[0],nod[1]);*/
                                                        /*printf("\nOri = %s
Nom = %s Tipo = %c",ori,nam,tipo);getch();*/
pos = no1( nam, ori, &tipcaj, &orden ) ;

if( pos != -1 ) {
        if( tipcaj == 'e' ) {
                ent = (I.ent+pos) ;
                for( i=0 ; i < ent->ndest ; i++ )
                        if( ( strcmp( *(ent->nom+i), nod[0] ) == NULL ) && ( strcmp( *(ent->des+i),
noc ) == NULL ) && ( *(ent->tipo+i) != 'c' ) ) {
                                *((I.ent+pos)->nom+i) = memr_char( *((I.ent+pos)->nom+i),
(strlen(nod[1]) + 1),  canod4z(*I.ent->nom)  ) ;
                                strcpy( *(ent->nom+i), nod[1] ) ;
                                break ;
                                }
                }

        else if( tipcaj == 's' ) {
                ent = (II.cs+orden)->sal+pos) ;
                for( i=0 ; i < ent->ndest ; i++ )
                        if ( strcmp( *(ent->nom+i), nod[0] ) == NULL ) && ( strcmp( *(ent->des+i),
noc ) == NULL ) && ( *(ent->tipo+i) != 'c' ) ) {
                                *((II.cs+orden)->sal+pos)->nom+i) = memr_char( *((II.cs+orden)-
>sal+pos)->nom+i), (strlen(nod[1]) + 1),  canod4z(*I.cs->sal->nom)  ) ;
                                strcpy( *(ent->nom+i), nod[1] ) ;
                                break ;
                                }
                }

        else if( tipcaj == 'c' ) {
                ent = (II.cc+orden)->sal+pos) ;
                for( i=0 ; i < ent->ndest ; i++ )
                        if ( strcmp( *(ent->nom+i), nod[0] ) == NULL ) && ( strcmp( *(ent->des+i),
noc ) == NULL ) && ( *(ent->tipo+i) != 'c' ) ) {
                                *((II.cc+orden)->sal+pos)->nom+i) = memr_char( *((II.cc+orden)-
>sal+pos)->nom+i), (strlen(nod[1]) + 1),  canod4z(*I.cc->sal->nom)  ) ;
                                strcpy( *(ent->nom+i), nod[1] ) ;
                                break ;
                                }
                }
        }

if( tipo == 'p' )
        for( i=0 ; i < I.nbloc ; i++ ) {
                co = &((II.buc+i)->co) ;
```

```
                for( j=0 ; j < co->norig ; j++ )
                        if( (strcmp( *(co->nom+j), nod[0] ) == NULL) && (strcmp( *(co->ori+j), noc
== NULL) ) {
                                *((l.buc+i)->eb.nom+j) = memr_char( *((l.buc+i)->eb.nom+j),
(strlen(nod[1]) +1), "canoda2(*l.buc->eb.nom)" ) ;
                                strcpy( *(co->nom+j), nod[1] ) ;
                                i = l.nbuc ;
                                break ;
                                }
                }
else if( (tipo == 'o') && (strcmp( ori, "cod" ) == NULL) ) {
        for( i=0 ; i < l.ninic ; i++ ) {
                ini = (l.ini+i) ;
                if( (strcmp( ini->caja, noc ) == NULL) && (strcmp( ini->inp, nod[0] ) == NULL) ) {
                        (l.ini+i)->inp = memr_char( (l.ini+i)->inp, (strlen(nod[1]) + 1), "canoda2(l.ini-
>inp)" ) ;
                        strcpy( ini->inp, nod[1] ) ;
                        break ;
                        }
                }
        }
}
```

```
#include "head.h"
#include <stdlib.h>
#include <stdio.h>
#include <dir.h>
#include <dos.h>
#include <graphics.h>

extern struct lectur l ;

/*-------------------------------------------------------------------*/
/*      ME51                          MENU DE PONER                  */
/*-------------------------------------------------------------------*/

int     me51( un, xymax, col )

struct unidad     *un     ;
int               xymax[ ];
struct color      col     ;

{

static int i=0    ;
int               k       ;
int               x       ;
int               y       ;
int               ncadna  ;
int               xy[8]   ;
int               flag3   ;
unsigned size     ;
char              noc[2][11];
char              nod[2][21];
char              tipcaj[2];
char              elemento[11];
char              id[11]  ;
char              name[10][16];
int               orden[2];
int               me516() ;
FILE              *fp     ;
char    *tex51[ ] = {
                  "Caja simp.",
                  "Caja comp.",
                  "Bloque IF",
                  "Bucle",
                  "Recursion",
                  "Iniciali.",
                  "Datos" } ;
char    *let51[ ] = {"s","c","F","B","R","I","D"} ;
int     pos51[ ] = { 5, 5, 8, 0, 0, 0, 0 } ;
char    *text[ ] = {" No se puede poner nombres distintos",
                    " a un mismo dato del exterior." } ;


k = 7 ;
mecoor( un, xymax, xy, k, 10, 2 ) ;

flag3 = 0 ;
size = imagesize( xy[0], xy[1], xy[4], xy[5] ) ;

if( coreleft( ) > ( size + 4000 ) ) {
        flag3 = 1 ;
        if( ( fp = tmpfile( ) ) == NULL )
```

```
                { closegraph( ) ; perror("error: me51: tmpfile: ") ; exit( 1 ) ; }
        mecoj( fp, xy, size ) ;
        }

menu( un, col, tex51, let51, pos51, xy, k ) ;

do      {
        y = 1 ;
        x = resp( un, xy, &i, k, 1, 51 ) ;

        switch( x ) {
                case -2 :
                case -1 :
                        y = 0 ;
                        break ;
                case 0 :
                case 1 :
                        if( me1ele( un, xymax, col, -1, -1, elemento ) == -1 ||
                           meid( un, xymax, col, id ) == -1 ||
                          ( ncadna = mecad( un, xymax, col, name ) ) == -1 )
                                break ;
                        if( x == 0 )
                                poncaj( 's', elemento, id, ncadna, name, -1, -1 ) ;
                        else
                                poncaj( 'c', elemento, id, ncadna, name, -1, -1 ) ;
                        break ;
                case 2 :
                        ponbif( un, xymax, col, -1, -1 ) ;
                        break ;
                case 3 :
                        ponbuc( un, xymax, col, -1, -1 ) ;
                        break ;
                case 4 :
                        me514( un, xymax, col ) ;
                        break ;
                case 5 :
                        me515( un, xymax, col ) ;
                        break ;
                case 6 :
                        if( me516( un, xymax, col, noc, nod, tipcaj, orden ) != -1 )
                                if( me517( un, xymax, col, noc, nod, tipcaj, orden ) != -1 )
                                        if( me518( noc, nod, tipcaj, orden ) == -1 )
                                                aviso( un, xymax, col, 37, 2, text ) ;
                }
        }
while( y ) ;

if( flag3 == 1 ) {
        mepon( fp, xy, size ) ;
        fclose( fp ) ;
        }
else    {
        merell( xy, col ) ;
        }

return( x ) ;

}

/*-----------------------------------------------------------------------*/
```

```
/*    ME514      MENU DE PONER RECURSIONES                              */
/*-------------------------------------------------------------------------*/

me514( un, xymax, col )

struct unidad       *un       ;
int                 xymax[ ];
struct color        col       ;

{

char                noc[9]    ;
char                nor[31]   ;
char                *mem_char(int,char * );
struct cursi        *mem_cursi(int,char * );
struct cursi        *memr_cursi(struct cursi * ,int,char * );


if( me5141( un, xymax, col, noc ) == -1 )
        return ;
                                                                    /*printf("\nCaja =
%s",noc);*/
if( metexto( un, xymax, col, nor, 1 ) == -1 )
        return ;
                                                                    /*printf("\nTexto =
%s",nor);*/
if( l.nrecur == 0 )
        l.rec = mem_cursi( 1, "me514(l.rec)" ) ;
else
        l.rec = memr_cursi( l.rec, (l.nrecur+1), "me514(l.rec)" ) ;

(l.rec+l.nrecur)->caja = mem_char( (strlen(noc) + 1), "me514(l.rec->caja)" ) ;
(l.rec+l.nrecur)->tex = mem_char( (strlen(nor) + 1), "me514(l.rec->tex)" ) ;

strcpy( (l.rec+l.nrecur)->caja, noc ) ;
strcpy( (l.rec+l.nrecur)->tex, nor ) ;

l.nrecur++ ;

}


/*-------------------------------------------------------------------------*/
/*    METEXTO                    MENU DE TEXTO AUXILIAR                    */
/*-------------------------------------------------------------------------*/

int       metexto( un, xymax, col, nor, flag )

struct unidad       *un       ;
int                 xymax[ ];
struct color        col       ;
char                nor[ ]    ;
int                 flag      ;

{

int                 k         ;
int                 x         ;
int                 xy[8]     ;
int                 curxy[2];
int                 flag3     ;
```

```
unsigned size       ;
int                 lecad( );
void                *buf      ;
char      *textex1[ ] = {
                    "Texto  >" } ;
char      *textex2[ ] = {
                    "Cond.  >" } ;
char      *textex3[ ] = {
                    "Con.> if" } ;
char      *lettex[ ] = {" "} ;
int       postex[ ] = { 0 } ;
void                *mem_buf(unsigned,char * );


k = 1 ;
mecoor( un, xymax, xy, k, 39, 3 ) ;

flag3 = 0 ;
size = imagesize( xy[0], xy[1], xy[4], xy[5] ) ;

if( coreleft( ) > ( size + 1000 ) ) {
          flag3 = 1 ;
          buf = mem_buf( size, "metexto" ) ;
          getimage( xy[0], xy[1], xy[4], xy[5], buf ) ;
          }

do        {
          if( flag == 1 )
                    menu( un, col, textex1, lettex, postex, xy, k ) ;
          else if( flag == 2 )
                    menu( un, col, textex2, lettex, postex, xy, k ) ;
          else
                    menu( un, col, textex3, lettex, postex, xy, k ) ;

          curxy[0] = xy[0] + un->tw * 11 ;
          curxy[1] = xy[1] + un->th * 3 ;

          if( lecad( un, col, curxy, nor, 30 ) == -1 ) {
                    x = -1 ;
                    break ;
                    }
          if( strlen( nor ) > 0 ) {
                    x = 0 ;
                    break ;
                    }
          }
while( 1 ) ;
                                                                    /*printf("\nMetexto
devuelve = %d",x);*/
if( flag3 == 1 ) {
          putimage( xy[0], xy[1], buf, COPY_PUT ) ;
          free( buf ) ;
          }
else      {
          merell( xy, col ) ;
          }

return( x ) ;

}
```

```
/*-------------------------------------------------------------------*/
/*    ME5141              MENU DE CAJA SIN RECURSION                 */
/*-------------------------------------------------------------------*/

int       me5141( un, xymax, col, noc )

struct unidad     *un      ;
int               xymax[ ];
struct color      col      ;
char              noc[ ]   ;

{

int               k        ;
int               i        ;
int               j        ;
int               x        ;
int               y        ;
int               opc      ;
int               nele     ;
int               xy[8]    ;
int               flag3    ;
unsigned size     ;
FILE              *fp      ;
char              (*text)[11];
char      titulo[ ] = {"CAJA" } ;
union inkey       {
                  char     ch[2]    ;
                  int      i        ;
                  }        c        ;
char              (*mem_charvec(int,char * ))[11];


nele = l.ncajis + l.ncajic - l.nrecur ;
if( nele == 0 )
          return( -1 ) ;
text = mem_charvec( nele, "me5141(text)" ) ;
for( k=0,i=0 ; i < l.ncajis ; i++ ) {
          for( j=0 ; j < l.nrecur ; j++ )
                    if( strcmp( (l.rec+j)->caja, (l.cs+i)->id ) == NULL )
                              break ;
          if( j == l.nrecur ) {
                    strcpy( (text+k), (l.cs+i)->id ) ;
                    k++ ;
                    }
          }
for( i=0 ; i < l.ncajic ; i++ ) {
          for( j=0 ; j < l.nrecur ; j++ )
                    if( strcmp( (l.rec+j)->caja, (l.cc+i)->id ) == NULL )
                              break ;
          if( j == l.nrecur ) {
                    strcpy( (text+k), (l.cc+i)->id ) ;
                    k++ ;
                    }
          }

if( nele == 1 ) {
          strcpy( noc, (text+0) ) ;
          free( text ) ;
          return( 0 ) ;
```

```
        }

if( nele < 9 ) {
        opc = 0 ;
        k = nele + 1 ;
        mecoor( un, xymax, xy, k, 12, 3 ) ;
        k-- ;
        }
else    {
        opc = 1 ;
        mecoor( un, xymax, xy, 9, 12, 3 ) ;
        k = 0 ;
        }

flag3 = 0 ;
size = imagesize( xy[0], xy[1], xy[4], xy[5] ) ;

if( coreleft( ) > ( size + 3000 ) ) {
        flag3 = 1 ;
        if( ( fp = tmpfile( ) ) == NULL )
                { closegraph( ) ; perror("error: mecaja: tmpfile: ") ; exit( 1 ) ; }
        mecoj( fp, xy, size ) ;
        }

i=0 ;

do      {
        y = 0 ;
        mevar( un, col, text, titulo, xy, k, opc ) ;

        x = revar( un, xy, &i, k, opc ) ;

        switch( x ) {
                case -2 :
                        y=1 ;
                        i=0 ;
                        k -= 8 ;
                        if( k <= 0 ) {
                                k=0 ;
                                opc = 1 ;
                                }
                        else
                                opc = 2 ;
                        break ;
                case -3 :
                        y=1 ;
                        i=0 ;
                        k += 8 ;
                        if( k > nele - 8 ) {
                                k = nele - 8 ;
                                opc = 3 ;
                                }
                        else
                                opc = 2 ;
                case -1 :
                        break ;
                default :
                        if( opc == 0 )
                                strcpy( noc, *(text+x) ) ;
                        else
                                strcpy( noc, *(text+k+x) ) ;
```

```
                }
        }
while( y ) ;

if( flag3 == 1 ) {
        mepon( fp, xy, size ) ;
        fclose(fp);
        }
else    {
        mereil( xy, col ) ;
        }

free( text ) ;
return( x ) ;

}

/*-------------------------------------------------------------------*/
/*    ME516                MENU DE CAJA ORIGEN                       */
/*-------------------------------------------------------------------*/

int     me516( un, xymax, col, noc, nod, tipcaj, orden )

struct unidad   *un     ;
int             xymax[] ;
struct color    col     ;
char            noc[][11];
char            nod[][21];
char            tipcaj[];
int             orden[] ;

{

int             k       ;
int             i       ;
int             j       ;
int             x       ;
int             y       ;
int             opc     ;
int             nele    ;
int             xy[8]   ;
int             flag3   ;
unsigned size   ;
FILE            *fp     ;
char            (*text)[11];
char    titulo[ ] = {"CAJA ORIG." } ;
union inkey     {
                char    ch[2]   ;
                int     i       ;
                }       c       ;
char            (*mem_charvec(int,char * ))[11];


nele = l.ncajis + l.ncajic + 1 ;
if( nele == 1 )
        return( -1 ) ;
text = mem_charvec( nele, "me516(text)" ) ;
for( i=0 ; i < l.ncajis ; i++ )
        strcpy(text+i,(l.cs+i)->id) ;
for( j=0 ; j < l.ncajic ; j++,i++ )
```

```
            strcpy(text+i,(l.cc+j)->id) ;
strcpy(text+i,"Exterior") ;

if( nele < 9 ) {
            opc = 0 ;
            k = nele + 1 ;
            mecoor( un, xymax, xy, k, 12, 3 ) ;
            k-- ;
            }
else        {
            opc = 1 ;
            mecoor( un, xymax, xy, 9, 12, 3 ) ;
            k = 0 ;
            }

flag3 = 0 ;
size = imagesize( xy[0], xy[1], xy[4], xy[5] ) ;

if( coreleft( ) > ( size + 3000 ) ) {
            flag3 = 1 ;
            if( ( fp = tmpfile( ) ) == NULL )
                        { closegraph( ) ; perror("error: me516: tmpfile: ") ; exit( 1 ) ; }
            mecoj( fp, xy, size ) ;
            }

i=0 ;

do          {
            y = 0 ;
            mevar( un, col, text, titulo, xy, k, opc ) ;

            x = revar( un, xy, &i, k, opc ) ;

            switch( x ) {
                        case -2 :
                                    y=1 ;
                                    i=0 ;
                                    k -= 8 ;
                                    if( k <= 0 ) {
                                                k=0 ;
                                                opc = 1 ;
                                                }
                                    else
                                                opc = 2 ;
                                    break ;
                        case -3 :
                                    y=1 ;
                                    i=0 ;
                                    k += 8 ;
                                    if( k > nele - 8 ) {
                                                k = nele - 8 ;
                                                opc = 3 ;
                                                }
                                    else
                                                opc = 2 ;
                        case -1 :
                                    break ;
                        default :
                                    if( opc == 0 )
                                                strcpy( noc[0], *(text+x) ) ;
                                    else
```

```
                                        strcpy( noc[0], *(text+k+x) ) ;
                                if( ( x = datori( un, xymax, col, noc[0], nod[0], tipcaj, orden ) ) == -1 )
                                        y = 1 ;
                        }
                }
while( y ) ;

if( flag3 == 1 ) {
        mepon( fp, xy, size ) ;
        fclose(fp);
        }
else    {
        merell( xy, col ) ;
        }

free( text ) ;
return( x ) ;

}

/*-------------------------------------------------------------------*/
/*    DATORI              MENU DE DATOS DE ORIGEN                    */
/*-------------------------------------------------------------------*/

int     datori( un, xymax, col, noc, nod, tipcaj, orden )

struct unidad   *un     ;
int             xymax[] ;
struct color    col     ;
char            noc[]   ;
char            nod[]   ;
char            *tipcaj ;
int             *orden  ;

{

int             k       ;
int             i       ;
int             x       ;
int             y       ;
int             opc     ;
int             nele    ;
int             xy[8]   ;
int             flag3   ;
unsigned size   ;
FILE            *fp     ;
char            *(*texi);
char            (*text)[11];
struct cajita   *caj    ;
char    aux[ ] = { " < otro >" } ;
char    titulo[ ] = {"DATO ORIG."} ;
union inkey     {
                char    ch[2]   ;
                int     i       ;
                }       c       ;
char            *(*mem_charp(int,char * ));
char            (*mem_charvec(int,char * ))[11];


ncl( noc, tipcaj, orden ) ;
```

```
if( *tipcaj == 'e' ) {
        texi = mem_charp( (l.nentra+1), "datori(texi)" ) ;
        for( i=0 ; i < l.nentra ; i++ )
                *(texi+i+1) = (l.ent+i)->nam ;
        }
else if( *tipcaj == 's' ) {
        caj = (l.cs+ *orden) ;
        texi = mem_charp( (caj->nsalid+1), "datori(texi)" ) ;
        for( i=0 ; i < caj->nsalid ; i++ )
                *(texi+i+1) = (caj->sal+i)->nam ;
        }
else    {
        caj = (l.cc+ *orden) ;
        texi = mem_charp( (caj->nsalid+1), "datori(texi)" ) ;
        for( i=0 ; i < caj->nsalid ; i++ )
                *(texi+i+1) = (caj->sal+i)->nam ;
        }
nele = i + 1 ;
if( nele == 1 ) {
        free( texi ) ;
        if( menoda( un, xymax, col, nod, 1 ) == -1 )
                return( -1 ) ;
        return( 0 ) ;
        }
*(texi) = aux ;
text = mem_charvec( nele, "datori(text)" ) ;
for( i=0 ; i < nele ; i++ )
        strncpy( *(text+i), *(texi+i), 10 ) ;

if( nele < 9 ) {
        opc = 0 ;
        k = nele + 1 ;
        mecoor( un, xymax, xy, k, 10, 4 ) ;
        k-- ;
        }
else    {
        opc = 1 ;
        mecoor( un, xymax, xy, 9, 10, 4 ) ;
        k = 0 ;
        }

flag3 = 0 ;
size = imagesize( xy[0], xy[1], xy[4], xy[5] ) ;

if( coreleft( ) > ( size + 2000 ) ) {
        flag3 = 1 ;
        if( ( fp = tmpfile( ) ) == NULL )
                { closegraph( ) ; perror("error: datori: tmpfile: ") ; exit( 1 ) ; }
        mecoj( fp, xy, size ) ;
        }

i=0 ;

do      {
        y = 0 ;
        mevar( un, col, text, titulo, xy, k, opc ) ;

        x = revar( un, xy, &i, k, opc ) ;

        switch( x ) {
                case -2 :
```

```
                        y=1 ;
                        i=0 ;
                        k -= 8 ;
                        if( k <= 0 ) {
                                k=0 ;
                                opc = 1 ;
                                }
                        else
                                opc = 2 ;
                        break ;
                case -3 :
                        y=1 ;
                        i=0 ;
                        k += 8 ;
                        if( k > nele - 8 ) {
                                k = nele - 8 ;
                                opc = 3 ;
                                }
                        else
                                opc = 2 ;
                case -1 :
                        break ;
                case 0 :
                        if( menoda( un, xymax, col, nod, 1 ) == -1 )
                                y=1 ;
                        break ;
                default :
                        if( opc == 0 )
                                strcpy( nod, *(texi+x) ) ;
                        else
                                strcpy( nod, *(texi+k+x) ) ;
                }
        }
while( y );

if( flag3 == 1 ) {
        mepon( fp, xy, size ) ;
        fclose(fp);
        }
else    {
        merell( xy, col ) ;
        }

free( text ) ;
free( texi ) ;
return( x ) ;

}


/*-------------------------------------------------------------------*/
/*    ME517                MENU DE CAJA DESTINO                      */
/*-------------------------------------------------------------------*/

int     me517( un, xymax, col, noc, nod, tipcaj, orden )

struct unidad   *un     ;
int             xymax[ ];
struct color    col     ;
char            noc[][11];
char            nod[][21];
```

```
char            tipcaj[];
int             orden[]  ;

{

int             k       ;
int             i       ;
int             j       ;
int             x       ;
int             y       ;
int             opc     ;
int             nele    ;
int             xy[8]   ;
int             flag3   ;
unsigned size   ;
FILE            *fp     ;
char            (*text)[11];
char    titulo[ ] = {"CAJA DESTINO" } ;
union inkey     {
                char    ch[2]   ;
                int     i       ;
                }       c       ;
char            (*mem_charvec(int,char * ))[11];
char            (*memr_charvec(char (*)[11] ,int,char * ))[11];


x = ph0( noc[0] ) ;
text = mem_charvec( 1, "me517(text)" ) ;
for( i=0,j=0 ; j < l.ncajis ; j++ )
        if( (l.cs+j)->pohor > x ) {
                strcpy(*(text+i++),(l.cs+j)->id) ;
                text = memr_charvec( text, (i+1), "me517(text)" ) ;
                }
for( j=0 ; j < l.ncajic ; j++ )
        if( (l.cc+j)->pohor > x ) {
                strcpy(*(text+i++),(l.cc+j)->id) ;
                text = memr_charvec( text, (i+1), "me517(text)" ) ;
                }
if( strcmp( noc[0], "Exterior" ) != NULL )
        strcpy(*(text+i++),"Exterior") ;
if( i==1 ) {
        strcpy( noc[1], *(text) ) ;
        free( text ) ;
        return( datdes( un, xymax, col, noc[1], nod[1], (tipcaj+1), (orden+1) ) ) ;
        }
nele = i ;

if( nele < 9 ) {
        opc = 0 ;
        k = nele + 1 ;
        mecoor( un, xymax, xy, k, 12, 3 ) ;
        k-- ;
        }
else    {
        opc = 1 ;
        mecoor( un, xymax, xy, 9, 12, 3 ) ;
        k = 0 ;
        }

flag3 = 0 ;
size = imagesize( xy[0], xy[1], xy[4], xy[5] ) ;
```

```
if( coreleft( ) > ( size + 3000 ) ) {
        flag3 = 1 ;
        if( ( fp = tmpfile( ) ) = = NULL )
                { closegraph( ) ; perror("error: me517: tmpfile: ") ; exit( 1 ) ; }
        mecoj( fp, xy, size ) ;
        }

i=0 ;

do      {
        y = 0 ;
        mevar( un, col, text, titulo, xy, k, opc ) ;

        x = revar( un, xy, &i, k, opc ) ;

        switch( x ) {
                case -2 :
                        y=1 ;
                        i=0 ;
                        k -= 8 ;
                        if( k < = 0 ) {
                                k=0 ;
                                opc = 1 ;
                                }
                        else
                                opc = 2 ;
                        break ;
                case -3 :
                        y=1 ;
                        i=0 ;
                        k + = 8 ;
                        if( k > nele - 8 ) {
                                k = nele - 8 ;
                                opc = 3 ;
                                }
                        else
                                opc = 2 ;
                case -1 :
                        break ;
                default :
                        if( opc = = 0 )
                                strcpy( noc[1], *(text+x) ) ;
                        else
                                strcpy( noc[1], *(text+k+x) ) ;
                        if( ( x = datdes( un, xymax, col, noc[1], nod[1], (tipcaj+1), (orden+1) ) ) = = -1
)
                                y = 1 ;
                }
        }
while( y ) ;

if( flag3 = =1 ) {
        mepon( fp, xy, size ) ;
        fclose(fp);
        }
else    {
        merell( xy, col ) ;
        }

free( text ) ;
```

```
return( x ) ;

}

/*-------------------------------------------------------------------*/
/*    DATDES                MENU DE DATOS DE DESTINO                 */
/*-------------------------------------------------------------------*/

int       datdes( un, xymax, col, noc, nod, tipcaj, orden )

struct unidad     *un      ;
int               xymax[]  ;
struct color      col      ;
char              noc[]    ;
char              nod[]    ;
char              *tipcaj  ;
int               *orden   ;

{

int               k        ;
int               i        ;
int               x        ;
int               y        ;
int               opc      ;
int               nele     ;
int               xy[8]    ;
int               flag3    ;
unsigned size     ;
FILE              *fp      ;
char              *(*texi);
char              (*text)[11];
struct cajita     *caj     ;
char      aux[ ] = { " < otro >" } ;
char      titulo[ ] = {"DATO DEST."} ;
union inkey       {
                  char      ch[2]    ;
                  int       i        ;
                  }         c        ;
char              *(*mem_charp(int,char * ));
char              (*mem_charvec(int,char * ))[11];


ncl( noc, tipcaj, orden ) ;
if( *tipcaj == 's' ) {
          texi = mem_charp( (l.nsalid+1), "datdes(texi)" ) ;
          for( i=0 ; i < l.nsalid ; i++ )
                    *(texi+i+1) = (l.sal+i)->nam ;
          }
else if( *tipcaj == 'e' ) {
          caj = (l.cs+ *orden) ;
          texi = mem_charp( (caj->nentra+1), "datdes(texi)" ) ;
          for( i=0 ; i < caj->nentra ; i++ )
                    *(texi+i+1) = (caj->ent+i)->nam ;
          }
else      {
          caj = (l.cc+ *orden) ;
          texi = mem_charp( (caj->nentra+1), "datdes(texi)" ) ;
          for( i=0 ; i < caj->nentra ; i++ )
                    *(texi+i+1) = (caj->ent+i)->nam ;
```

```
        }
nele = i + 1 ;
if( nele == 1 ) {
        free( texi ) ;
        if( menoda( un, xymax, col, nod, 1 ) == -1 )
                return( -1 ) ;
        return( 0 ) ;
        }
*(texi) = aux ;
text = mem_charvec( nele, "datdes(text)" ) ;
for( i=0 ; i < nele ; i++ )
        strncpy( *(text+i), *(texi+i), 10 ) ;

if( nele < 9 ) {
        opc = 0 ;
        k = nele + 1 ;
        mecoor( un, xymax, xy, k, 10, 4 ) ;
        k-- ;
        }
else    {
        opc = 1 ;
        mecoor( un, xymax, xy, 9, 10, 4 ) ;
        k = 0 ;
        }

flag3 = 0 ;
size = imagesize( xy[0], xy[1], xy[4], xy[5] ) ;

if( coreleft( ) > ( size + 2000 ) ) {
        flag3 = 1 ;
        if( ( fp = tmpfile( ) ) == NULL )
                { closegraph( ) ; perror("error: datdes: tmpfile: ") ; exit( 1 ) ; }
        mecoj( fp, xy, size ) ;
        }

i=0 ;

do      {
        y = 0 ;
        mevar( un, col, text, titulo, xy, k, opc ) ;

        x = revar( un, xy, &i, k, opc ) ;

        switch( x ) {
                case -2 :
                        y=1 ;
                        i=0 ;
                        k -= 8 ;
                        if( k <= 0 ) {
                                k=0 ;
                                opc = 1 ;
                                }
                        else
                                opc = 2 ;
                        break ;
                case -3 :
                        y=1 ;
                        i=0 ;
                        k += 8 ;
                        if( k > nele - 8 ) {
                                k = nele - 8 ;
```

```
                                        opc = 3 ;
                                        }
                                else
                                        opc = 2 ;
                        case -1 :
                                break ;
                        case 0 :
                                if( menoda( un, xymax, col, nod, 1 ) == -1 )
                                        y=1 ;
                                break ;
                        default :
                                if( opc == 0 )
                                        strcpy( nod, *(texi+x) ) ;
                                else
                                        strcpy( nod, *(texi+k+x) ) ;
                        }
                }
while( y ) ;

if( flag3 == 1 ) {
                mepon( fp, xy, size ) ;
                fclose(fp);
                }
else            {
                merell( xy, col ) ;
                }

free( text ) ;
free( texi ) ;
return( x ) ;

}

/*-------------------------------------------------------------------*/
/*      MENODA                          VENTANA DE NOMBRE DE DATO    */
/*-------------------------------------------------------------------*/

int     menoda( un, xymax, col, nod, flag )

struct unidad   *un     ;
int             xymax[ ];
struct color    col     ;
char            nod[21] ;
int             flag    ;

{

int             x       ;
int             xy[8]   ;
int             curxy[2];
int             flag3   ;
unsigned size   ;
int             lecad( );
void            *buf    ;
char    *tex522_1[ ] = { " Nombre  >" } ;
char    *tex522_2[ ] = { " Valor   >" } ;
char    *let522[ ] = {" "} ;
int     pos522[ ] = { 0 } ;
void            *mem_buf(unsigned,char * );
```

```
mecoor( un, xymax, xy, 1, 31, 3 ) ;

flag3 = 0 ;
size = imagesize( xy[0], xy[1], xy[4], xy[5] ) ;

if( coreleft( ) > ( size + 1000 ) ) {
        flag3 = 1 ;
        buf = mem_buf( size, "menoda" ) ;
        getimage( xy[0], xy[1], xy[4], xy[5], buf ) ;
        }

if( flag == 1 )
        menu( un, col, tex522_1, let522, pos522, xy, 1 ) ;
else
        menu( un, col, tex522_2, let522, pos522, xy, 1 ) ;

curxy[0] = xy[0] + un->tw * 13 ;
curxy[1] = xy[1] + un->th * 3 ;

x = lecad( un, col, curxy, nod, 20 ) ;

if( flag3 == 1 ) {
        putimage( xy[0], xy[1], buf, COPY_PUT ) ;
        free( buf ) ;
        }
else    {
        setcolor( col.fond ) ;
        setfillstyle( SOLID_FILL, col.fond ) ;
        fillpoly( 4, xy ) ;
        }

return( x ) ;

}


/*-------------------------------------------------------------------*/
/*    ME518                AÑADIR UN DATO                            */
/*-------------------------------------------------------------------*/

int     me518( noc, nod, tipcaj, orden )

char            noc[][11];
char            nod[][21];
char            tipcaj[];
int             orden[] ;

{

int             pos[2]  ;
int             i       ;
struct entrad   *ent    ;
struct salida   *sal    ;
int             ni2()   ;
int             no2()   ;

if( strcmp( noc[0], "Exterior" ) == NULL )
        strcpy( noc[0], "ext" ) ;
if( strcmp( noc[1], "Exterior" ) == NULL )
```

```
        strcpy( noc[1], "ext" ) ;

pos[0] = no2( nod[0], tipcaj[0], orden[0] ) ;
pos[1] = ni2( nod[1], tipcaj[1], orden[1] ) ;

if( pos[0] = = -1  && tipcaj[0] = = 'e'  && pos[1] != -1 ) {
        if( tipcaj[1] = = 'e' )
                sal = (l.sal + pos[1]) ;
        else if( tipcaj[1] = = 's' )
                sal = ((l.cs + orden[1])->ent + pos[1]) ;
        else if( tipcaj[1] = = 'c' )
                sal = ((l.cc + orden[1])->ent + pos[1]) ;

        for( i=0 ; i < sal->norig ; i++ )
                if( strcmp( *(sal->ori + i), "ext" ) = = NULL )
                        return( -1 ) ;
        }

if( pos[1] = = -1  && tipcaj[1] = = 'e'  && pos[0] != -1 ) {
        if( tipcaj[0] = = 'e' )
                ent = (l.ent + pos[0]) ;
        else if( tipcaj[0] = = 's' )
                ent = ((l.cs + orden[0])->sal + pos[0]) ;
        else if( tipcaj[0] = = 'c' )
                ent = ((l.cc + orden[0])->sal + pos[0]) ;

        for( i=0 ; i < ent->ndest ; i++ )
                if( strcmp( *(ent->des + i), "ext" ) = = NULL )
                        return( -1 ) ;
        }

pondes( noc[1], nod, tipcaj[0], orden[0], pos[0], 'i' ) ;
ponori( noc[0], nod, tipcaj[1], orden[1], pos[1], 'o' ) ;

}


/*-------------------------------------------------------------------*/
/*    ME515      MENU DE PONER INICIALICIACIONES                     */
/*-------------------------------------------------------------------*/

me515( un, xymax, col )

struct unidad       *un     ;
int                 xymax[] ;
struct color        col     ;

{

int                 x       ;
int                 i       ;
int                 orden   ;
int                 pos     ;
char                tipcaj  ;
char                noc[11] ;
char                nocaux[11];
char                nod[2][21];
char                valor[21];
int                 me5151();
int                 datdes();
int                 ph0()   ;
```

```
char        *text[]={ "<- Ese dato ya está inicializado ->",
                      "  Si desea cambiar algo, vaya a la",
                      "opción \"cambiar\" del menú modificar." } ;
char                *mem_char(int,char * );
struct inici        *mem_inici(int,char * );
struct inici        *memr_inici(struct inici * ,int,char * );


if( me5151( un, xymax, col, noc ) == -1 )
        return ;

if( datdes( un, xymax, col, noc, nod[1], &tipcaj, &orden ) == -1 )
        return ;

for( i=0 ; i < l.ninic ; i++ )
        if( strcmp( (l.ini+i)->caja, noc ) == NULL &&
            strcmp( (l.ini+i)->inp, nod[1] ) == NULL ) {
                aviso( un, xymax, col, 36, 3, text ) ;
                return ;
                }

if( menoda( un, xymax, col, valor, 2 ) == -1 )
        return ;

if( mehuini( un, xymax, col, noc, nocaux ) == -1 )
        return ;

if( l.ninic == 0 )
        l.ini = mem_inici( 1, "me515(ini)" ) ;
else
        l.ini = memr_inici( l.ini, (l.ninic+1), "me515(ini)" ) ;

(l.ini+l.ninic)->caja = mem_char( (strlen( noc ) +1), "me515(*ini)" ) ;
(l.ini+l.ninic)->inp = mem_char( (strlen( nod[1] ) +1), "me515(*ini)" ) ;
(l.ini+l.ninic)->val = mem_char( (strlen( valor ) +1), "me515(*ini)" ) ;
(l.ini+l.ninic)->anca = mem_char( (strlen( nocaux ) +1), "me515(*ini)" ) ;

strcpy( (l.ini+l.ninic)->caja, noc ) ;
strcpy( (l.ini+l.ninic)->inp, nod[1] ) ;
strcpy( (l.ini+l.ninic)->val, valor ) ;
strcpy( (l.ini+l.ninic)->anca, nocaux ) ;
l.ninic++ ;

strcpy( nod[0], "x" ) ;
pos = ni2( nod[1], tipcaj, orden ) ;
ponori( "cod", nod, tipcaj, orden, pos, 'o' ) ;

}


/*-------------------------------------------------------------------*/
/*    ME5151    MENU DE CAJAS PARA INICIALICIACION                   */
/*-------------------------------------------------------------------*/

int         me5151( un, xymax, col, noc )

struct unidad       *un     ;
int                 xymax[] ;
struct color        col     ;
char                noc[]   ;
```

```
{

int                 k       ;
int                 i       ;
int                 j       ;
int                 x       ;
int                 y       ;
int                 opc     ;
int                 nele    ;
int                 xy[8]   ;
int                 flag3   ;
unsigned size       ;
FILE                *fp     ;
char                (*text)[11];
char      titulo[ ] = {"CAJA" } ;
union inkey         {
                    char    ch[2]   ;
                    int     i       ;
                    }       c       ;
char                (*mem_charvec(int,char * ))[11];


nele = l.ncajis + l.ncajic ;
if( nele = = 0 )
          return( -1 ) ;
else if( nele = = 1 ) {
          if( l.ncajis = = 1 )
                    strcpy( noc, l.cs->id ) ;
          else
                    strcpy( noc, l.cc->id ) ;
          return( 0 ) ;
          }
text = mem_charvec( nele, "me5151(text)" ) ;
for( i=0 ; i < l.ncajis ; i++ )
          strcpy( *(text+i), (l.cs+i)->id ) ;
for( j=0 ; j < l.ncajic ; j++ ,i++ )
          strcpy( *(text+i), (l.cc+j)->id ) ;

if( nele < 9 ) {
          opc = 0 ;
          k = nele + 1 ;
          mecoor( un, xymax, xy, k, 12, 3 ) ;
          k-- ;
          }
else      {
          opc = 1 ;
          mecoor( un, xymax, xy, 9, 12, 3 ) ;
          k = 0 ;
          }

flag3 = 0 ;
size = imagesize( xy[0], xy[1], xy[4], xy[5] ) ;

if( coreleft( ) > ( size + 3000 ) ) {
          flag3 = 1 ;
          if( ( fp = tmpfile( ) ) = = NULL )
                    { closegraph( ) ; perror("error: me516: tmpfile: ") ; exit( 1 ) ; }
          mecoj( fp, xy, size ) ;
          }

i=0 ;
```

```
do      {
        y = 0 ;
        mevar( un, col, text, titulo, xy, k, opc ) ;

        x = revar( un, xy, &i, k, opc ) ;

        switch( x ) {
                case -2 :
                        y=1 ;
                        i=0 ;
                        k -= 8 ;
                        if( k <= 0 ) {
                                k=0 ;
                                opc = 1 ;
                                }
                        else
                                opc = 2 ;
                        break ;
                case -3 :
                        y=1 ;
                        i=0 ;
                        k += 8 ;
                        if( k > nele - 8 ) {
                                k = nele - 8 ;
                                opc = 3 ;
                                }
                        else
                                opc = 2 ;
                case -1 :
                        break ;
                default :
                        if( opc == 0 )
                                strcpy( noc, *(text+x) ) ;
                        else
                                strcpy( noc, *(text+k+x) ) ;
                }
        }
while( y ) ;

if( flag3 == 1 ) {
        mepon( fp, xy, size ) ;
        fclose(fp);
        }
else    {
        merell( xy, col ) ;
        }

free( text ) ;
return( x ) ;

}


/*-------------------------------------------------------------------*/
/*    MEHUINI    MENU DE CAJAS PARA HUECO DE INICIALICIACION       */
/*-------------------------------------------------------------------*/

int     mehuini( un, xymax, col, noc1, noc0 )

struct unidad     *un     ;
```

```
int             xymax[] ;
struct color    col     ;
char            noc1[]  ;
char            noc0[]  ;

{

int             k       ;
int             i       ;
int             x       ;
int             y       ;
int             opc     ;
int             nele    ;
int             xy[8]   ;
int             flag3   ;
int             pv0()   ;
int             ph0()   ;
unsigned size   ;
FILE            *fp     ;
char            (*text)[11];
char    titulo[ ] = {"ANTES DE:" } ;
struct cajita   *cji    ;
union inkey     {
                char    ch[2]   ;
                int     i       ;
                }       c       ;
char            (*mem_charvec(int,char * ))[11];


nele = ph0( noc1 ) + 1 ;
if( nele == 1 ) {
        strcpy( noc0, noc1 ) ;
        return( 0 ) ;
        }
text = mem_charvec( nele, "me5151(text)" ) ;
for( i=0 ; i < l.ncajis ; i++ ) {
        cji = (l.cs+i) ;
        if( cji->pohor < (nele - 1) )
                if( cji->pover < pv0( noc1 ) &&
                  ( ((x = strlen( *(text+cji->pohor) ) ) > 0 && pv0( *(text+cji->pohor) ) < cji->pover )
|| x == 0 ) )
                        strcpy( *(text+cji->pohor), cji->id ) ;
        }
for( i=0 ; i < l.ncajic ; i++ ) {
        cji = (l.cc+i) ;
        if( cji->pohor < (nele - 1) )
                if( cji->pover < pv0( noc1 ) &&
                  ( ((x = strlen( *(text+cji->pohor) ) ) > 0 && pv0( *(text+cji->pohor) ) < cji->pover )
|| x == 0 ) )
                        strcpy( *(text+cji->pohor), cji->id ) ;
        }
strcpy( *(text+nele-1), noc1 ) ;

if( nele < 9 ) {
        opc = 0 ;
        k = nele + 1 ;
        mecoor( un, xymax, xy, k, 12, 3 ) ;
        k-- ;
        }
else    {
        opc = 1 ;
```

```
        mecoor( un, xymax, xy, 9, 12, 3 ) ;
        k = 0 ;
        }

flag3 = 0 ;
size = imagesize( xy[0], xy[1], xy[4], xy[5] ) ;

if( coreleft( ) > ( size + 3000 ) ) {
        flag3 = 1 ;
        if( ( fp = tmpfile( ) ) == NULL )
                { closegraph( ) ; perror("error: me516: tmpfile: ") ; exit( 1 ) ; }
        mecoj( fp, xy, size ) ;
        }

i=0 ;

do      {
        y = 0 ;
        mevar( un, col, text, titulo, xy, k, opc ) ;

        x = revar( un, xy, &i, k, opc ) ;

        switch( x ) {
                case -2 :
                        y=1 ;
                        i=0 ;
                        k -= 8 ;
                        if( k <= 0 ) {
                                k=0 ;
                                opc = 1 ;
                                }
                        else
                                opc = 2 ;
                        break ;
                case -3 :
                        y=1 ;
                        i=0 ;
                        k += 8 ;
                        if( k > nele - 8 ) {
                                k = nele - 8 ;
                                opc = 3 ;
                                }
                        else
                                opc = 2 ;
                case -1 :
                        break ;
                default :
                        if( opc == 0 )
                                strcpy( noc0, *(text+x) ) ;
                        else
                                strcpy( noc0, *(text+k+x) ) ;
                }
        }
while( y ) ;

if( flag3 == 1 ) {
        mepon( fp, xy, size ) ;
        fclose(fp);
        }
else    {
        merell( xy, col ) ;
```

```
        }

free( text ) ;
return( x ) ;

}
```

```
#include "head.h"
#include <stdlib.h>
#include <graphics.h>
#define ESC 27
#define CR 13
#define ARRIBA 72
#define ABAJO 80
#define AvPg 81
#define RePg 73

/*-------------------------------------------------------------------*/
/*    MENU                    VISUALIZACION DE MENU                  */
/*-------------------------------------------------------------------*/

int       menu( un, col, texto, letra, pos, xy, k )

struct unidad     *un     ;
struct color      col     ;
char              *texto[];
char              *letra[];
int               pos[ ]  ;
int               xy[ ]   ;
int               k       ;

{

int               i       ;


setcolor( col.reve ) ;
setfillstyle( SOLID_FILL, col.fove ) ;
setlinestyle( SOLID_LINE, 0, NORM_WIDTH ) ;
fillpoly( 4, xy ) ;
rectangle((xy[0]+un->tw/2),(xy[1]+un->th/2),(xy[4]-un->tw/2),(xy[5]-un->th/2)) ;
settextjustify( LEFT_TEXT, BOTTOM_TEXT ) ;
settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 ) ;

setcolor( col.teve ) ;

for( i=0 ; i<k ; i++ )
        outtextxy( (xy[0]+(2*un->tw)), (xy[1]+((6+5*i)*un->th)/2), texto[i] ) ;

setcolor( col.inve ) ;

for( i=0 ; i<k ; i++ )
        outtextxy( (xy[0]+((2+pos[i])*un->tw)), (xy[1]+((6+5*i)*un->th)/2), letra[i] ) ;

}


/*-------------------------------------------------------------------*/
/*    RESP                    RESPUESTA DEL MENU                     */
/*-------------------------------------------------------------------*/

int       resp( un, xy, i, k, int1, int2 )

struct unidad     *un     ;
int               xy[ ]   ;
int               *i      ;
int               k       ;
int               int1    ;
```

```
int             int2    ;

{

int             n       ;
void            *buf    ;
void            *mem_buf(unsigned,char * );
union inkey     {
                char    ch[2]   ;
                int     i       ;
                }       c       ;


buf = mem_buf( imagesize( (xy[0]+un->tw), 0, (xy[4]-un->tw), (5*un->th/2) ), "resp" ) ;
getimage( (xy[0]+un->tw), (xy[1]+(2+5*(*i))*un->th/2), (xy[4]-un->tw), (xy[1]+(7+5*(*i))*un->th/2),
buf ) ;
putimage( (xy[0]+un->tw), (xy[1]+(2+5*(*i))*un->tw/2), buf, NOT_PUT ) ;

do      {
        while( !bioskey( 1 ) ) ;
        c.i = bioskey( 0 ) ;

        putimage( (xy[0]+un->tw), (xy[1]+(2+5*(*i))*un->th/2), buf, COPY_PUT ) ;

        if( ( n = clave( c.ch, int1, int2 ) ) != -1 ) {
                free( buf ) ;
                return( n ) ;
                }

        if( c.ch[0] ) {
                switch( c.ch[0] ) {

                        case CR :
                                free( buf ) ;
                                return( *i ) ;
                        case ESC :
                                free( buf ) ;
                                return( -1 ) ;
                        }
                }
        else    {
                switch( c.ch[1] ) {

                        case ARRIBA :
                                (*i)-- ;
                                break ;
                        case ABAJO :
                                (*i)++ ;
                        }
                }

        if( *i == k )
                *i=0 ;
        if( *i<0 )
                *i=k-1 ;

        getimage( (xy[0]+un->tw), (xy[1]+(2+5*(*i))*un->th/2), (xy[4]-un->tw), (xy[1]+(7+5*(*i))*un-
>tw/2), buf ) ;
        putimage( (xy[0]+un->tw), (xy[1]+(2+5*(*i))*un->th/2), buf, NOT_PUT ) ;

        }
```

```
while( 1 ) ;

}

/*-------------------------------------------------------------------*/
/*     MEVAR      VISUALIZACION DE MENU VARIABLE                     */
/*-------------------------------------------------------------------*/

int         mevar( un, col, text, titulo, xy, k, opc )

struct unidad       *un     ;
struct color        col     ;
char                (*text)[11];
char                titulo[];
int                 xy[ ]   ;
int                 k       ;
int                 opc     ;

{

int                 i       ;


setcolor( col.reve ) ;
setfillstyle( SOLID_FILL, col.fove ) ;
setlinestyle( SOLID_LINE, 0, NORM_WIDTH ) ;
fillpoly( 4, xy ) ;
if( opc == 0 )
        rectangle((xy[0]+un->tw/2),(xy[1]+(5*un->th)/2),(xy[4]-un->tw/2),(xy[5]-un->th/2)) ;
else
        rectangle((xy[0]+un->tw/2),(xy[1]+(5*un->th)/2),(xy[4]-un->tw/2),(xy[5]-(5*un->th)/4)) ;
settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 ) ;

settextjustify( CENTER_TEXT, BOTTOM_TEXT ) ;
outtextxy( ((xy[0]+xy[2])/2), (xy[1]+2*un->th), titulo ) ;

settextjustify( LEFT_TEXT, BOTTOM_TEXT ) ;

if( opc == 0 ) {
        setcolor( col.teve ) ;
        for( i=1 ; i<=k ; i++ )
                outtextxy( (xy[0]+(2*un->tw)), (xy[1]+((5+5*i)*un->th)/2), *(text+i-1) ) ;
        }
else    {
        switch( opc ) {

                case 1 :
                        outtextxy( (xy[2]-(5*un->tw)), xy[5],"AvPg") ;
                        break ;
                case 2 :
                        outtextxy( (xy[2]-(10*un->tw)), xy[5],"RePg/AvPg") ;
                        break ;
                case 3 :
                        outtextxy( (xy[2]-(5*un->tw)), xy[5],"RePg") ;
                }
        setcolor( col.teve ) ;
        for( i=1 ; i<9 ; i++,k++ )
                outtextxy( (xy[0]+(2*un->tw)), (xy[1]+((5+5*i)*un->th)/2), *(text+k) ) ;
        }
```

```
}

/*-------------------------------------------------------------------*/
/*    REVAR                RESPUESTA DEL MENU VARIABLE                */
/*-------------------------------------------------------------------*/

int       revar( un, xy, i, k, opc )

struct unidad     *un     ;
int               xy[ ]   ;
int               *i      ;
int               k       ;
int               opc     ;

{

int               n       ;
void              *buf    ;
void              *mem_buf(unsigned,char * );
union inkey       {
                  char    ch[2]   ;
                  int     i       ;
                  }       c       ;


if( opc != 0 )
        k = 8 ;

buf = mem_buf( imagesize( (xy[0]+un->tw), 0, (xy[4]-un->tw), (5*un->th/2) ), "revar" ) ;
getimage( (xy[0]+un->tw), (xy[1]+(6+5*(*i))*un->th/2), (xy[4]-un->tw), (xy[1]+(11+5*(*i))*un->th/2),
buf ) ;
putimage( (xy[0]+un->tw), (xy[1]+(6+5*(*i))*un->tw/2), buf, NOT_PUT ) ;

do      {
        while( !bioskey( 1 ) ) ;
        c.i = bioskey( 0 ) ;

        putimage( (xy[0]+un->tw), (xy[1]+(6+5*(*i))*un->th/2), buf, COPY_PUT ) ;

        if( c.ch[0] ) {
                switch( c.ch[0] ) {

                        case CR :
                                free( buf ) ;
                                return( *i ) ;
                        case ESC :
                                free( buf ) ;
                                return( -1 ) ;
                        }
                }
        else    {
                switch( c.ch[1] ) {

                        case ARRIBA :
                                (*i)-- ;
                                break ;
                        case ABAJO :
                                (*i)++ ;
                                break ;
```

```
                        case AvPg :
                                if( ( opc == 1 ) || ( opc == 2 ) ) {
                                        free( buf ) ;
                                        return( -3 ) ;
                                        }
                                break ;
                        case RePg :
                                if( ( opc == 2 ) || ( opc == 3 ) ) {
                                        free( buf ) ;
                                        return( -2 ) ;
                                        }
                        }
                }

        if( *i == k )
                *i=0 ;
        if( *i<0 )
                *i=k-1 ;

        getimage( (xy[0]+un->tw), (xy[1]+(6+5*(*i))*un->th/2), (xy[4]-un->tw),
(xy[1]+(11+5*(*i))*un->th/2), buf ) ;
        putimage( (xy[0]+un->tw), (xy[1]+(6+5*(*i))*un->th/2), buf, NOT_PUT ) ;

        }
while( 1 ) ;

}


/*-------------------------------------------------------------------*/
/*    AVISO                VENTANA DE AVISO                          */
/*-------------------------------------------------------------------*/

int     aviso( un, xymax, col, ancho, k, text )

struct unidad   *un     ;
int             xymax[ ];
struct color    col     ;
int             ancho   ;
int             k       ;
char            *text[ ];

{

int             i       ;
int             x       ;
int             y       ;
int             xy[8]   ;
int             flag3   ;
unsigned size   ;
void            *buf    ;
void            *mem_buf(unsigned,char * );


x = ( ancho + 4 ) * un->tw ;
y = ( ( ( k + 1 ) * 5 + 4 ) * un->th + 1 ) / 2 ;
xy[0] = ( xymax[0] - x ) / 2 ;
xy[1] = ( xymax[1] - y ) / 2 ;
xy[2] = xy[0] + x ;
xy[3] = xy[1] ;
xy[4] = xy[2] ;
```

```
xy[5] = xy[1] + y ;
xy[6] = xy[0] ;
xy[7] = xy[5] ;

flag3 = 0 ;
size = imagesize( xy[0], xy[1], xy[4], xy[5] ) ;

if( coreleft( ) > size ) {
        flag3 = 1 ;
        buf = mem_buf( size, "aviso" ) ;
        getimage( xy[0], xy[1], xy[4], xy[5], buf ) ;
        }

setcolor( col.reve ) ;
setfillstyle( SOLID_FILL, col.fove ) ;
setlinestyle( SOLID_LINE, 0, NORM_WIDTH ) ;
fillpoly( 4, xy ) ;
rectangle((xy[0]+un->tw/2),(xy[1]+(5*un->th)/2),(xy[4]-un->tw/2),(xy[5]-(5*un->th)/4)) ;
settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 ) ;

settextjustify( CENTER_TEXT, BOTTOM_TEXT ) ;
outtextxy( ((xy[0]+xy[2])/2), (xy[1]+2*un->th), "¡ ATENCION !" ) ;

settextjustify( RIGHT_TEXT, BOTTOM_TEXT ) ;
outtextxy( xy[2], xy[5],"Pulse cualquier tecla...") ;

settextjustify( LEFT_TEXT, BOTTOM_TEXT ) ;
setcolor( col.teve ) ;
k++ ;
for( i=1 ; i<k ; i++ )
        outtextxy( (xy[0]+(2*un->tw)), (xy[1]+((5+5*i)*un->th)/2), *(text+i-1) ) ;

bioskey( 0 ) ;

if( flag3 == 1 ) {
        putimage( xy[0], xy[1], buf, COPY_PUT ) ;
        free( buf ) ;
        }
else
        merell( xy, col ) ;

}
```

```
#include "head.h"
#include <stdlib.h>
#include <graphics.h>
#define ESC 27
#define DEL 8
#define CR 13

/*-------------------------------------------------------------------*/
/*      LEENT              LECTURA DE ENTERO DE VENTANA              */
/*-------------------------------------------------------------------*/

int     leent( un, col, curxy, entero, nele )

struct unidad     *un     ;
struct color      col     ;
int               curxy[ ];
int               *entero ;
int               nele    ;

{

int               ele     ;
int               xy[8]   ;
char              *dum    ;
char              *mem_char(int,char * );
union inkey       {
                  char    ch[2]   ;
                  int     i       ;
                  }       c       ;


dum = mem_char( (nele+1), "leent(dum)" ) ;

xy[0] = curxy[0] - ( un->tw + 1 ) / 2 ;
xy[1] = curxy[1] - un->th * 7 / 4 ;
xy[2] = xy[0] + un->tw * ( nele + 1 ) ;
xy[3] = xy[1] + un->th * 9 / 4 ;
setcolor( col.reve ) ;
setlinestyle( SOLID_LINE, 0, NORM_WIDTH ) ;
rectangle( xy[0], xy[1], xy[2], xy[3] ) ;

setcolor( col.tere ) ;
setfillstyle( SOLID_FILL, col.fove ) ;

ele = 0 ;

do      {
        while( !bioskey( 1 ) ) ;
        c.i = bioskey( 0 ) ;

        switch( c.ch[0] ) {

                case ESC :
                        free( dum ) ;
                        return( -1 ) ;
                case '0' :
                case '1' :
                case '2' :
                case '3' :
                case '4' :
                case '5' :
```

```
                case '6' :
                case '7' :
                case '8' :
                case '9' :
                        if( ele < nele ) {
                                *(dum+ele) = c.ch[0] ;
                                outtextxy( curxy[0], curxy[1], dum ) ;
                                ele++ ;
                                }
                        break ;
                case 0 :
                        if( c.ch[1] != 75 )
                                break ;
                case DEL :
                        if( ele != 0 ) {

                                ele-- ;
                                *(dum+ele) = '\0' ;
                                setcolor( col.fove ) ;

                                xy[0] = curxy[0] + un->tw * ele ;
                                xy[1] = curxy[1] ;
                                xy[2] = xy[0] ;
                                xy[3] = xy[1] - un->th ;
                                xy[4] = xy[0] + un->tw ;
                                xy[5] = xy[3] ;
                                xy[6] = xy[4] ;
                                xy[7] = xy[1] ;

                                fillpoly( 4, xy ) ;
                                setcolor( col.tere ) ;
                                }
                        break ;
                case CR :
                        *entero = atoi( dum ) ;
                        free( dum ) ;
                        return( 0 ) ;

                }
        }
while( 1 ) ;

}


/*-------------------------------------------------------------------*/
/*      LECAD           LECTURA DE CADENA DE VENTANA                 */
/*-------------------------------------------------------------------*/

int     lecad( un, col, curxy, cadena, nele )

struct unidad   *un     ;
struct color    col     ;
int             curxy[ ];
char            cadena[];
int             nele    ;

{

int             ele     ;
int             xy[8]   ;
```

```
union inkey     {
                char    ch[2]   ;
                int     i       ;
                }       c       ;


for( ele=0 ; ele < (nele+1) ; ele++ )
        cadena[ele] = '\0' ;

xy[0] = curxy[0] - ( un->tw +1 ) / 2 ;
xy[1] = curxy[1] - un->th * 2 ;
xy[2] = xy[0] + un->tw * ( nele + 1 ) ;
xy[3] = xy[1] + un->th * 5 / 2 ;
setcolor( col.reve ) ;
setlinestyle( SOLID_LINE, 0, NORM_WIDTH ) ;
rectangle( xy[0], xy[1], xy[2], xy[3] ) ;

setcolor( col.tere ) ;
setfillstyle( SOLID_FILL, col.fove ) ;

ele = 0 ;

do      {
        while( !bioskey( 1 ) ) ;
        c.i = bioskey( 0 ) ;

        switch( c.ch[0] ) {

                case ESC :
                        return( -1 ) ;
                case 0 :
                        if( c.ch[1] != 75 )
                                break ;
                case DEL :
                        if( ele != 0 ) {

                                ele-- ;
                                cadena[ele] = '\0' ;
                                setcolor( col.fove ) ;

                                xy[0] = curxy[0] + un->tw * ele ;
                                xy[1] = curxy[1] ;
                                xy[2] = xy[0] ;
                                xy[3] = xy[1] - un->th ;
                                xy[4] = xy[0] + un->tw ;
                                xy[5] = xy[3] ;
                                xy[6] = xy[4] ;
                                xy[7] = xy[1] ;

                                fillpoly( 4, xy ) ;
                                setcolor( col.tere ) ;
                                }
                        break ;
                case CR :
                        return( 0 ) ;
                default :
                        if( ele < nele ) {
                                cadena[ele] = c.ch[0] ;
                                outtextxy( curxy[0], curxy[1], cadena ) ;
                                ele++ ;
                                }
```

```
                }
        }
while( 1 ) ;

}

/*-------------------------------------------------------------------*/
/*      LECAMEN                 LECTURA DE CADENA DE VENTANA CON MENSAJE     */
/*-------------------------------------------------------------------*/

int     lecamen( un, col, curxy, cadena, nele, mens )

struct unidad   *un     ;
struct color    col     ;
int             curxy[] ;
char            cadena[];
int             nele    ;
char            mens[]  ;

{

int             i       ;
int             ele     ;
int             flag    ;
int             xy[8]   ;
char            cadaux[6];
union inkey     {
                char    ch[2]   ;
                int     i       ;
                }       c       ;


for( ele=0 ; ele < (nele+1) ; ele++ )
        cadena[ele] = '\0' ;

xy[0] = curxy[0] - ( un->tw +1 ) / 2 ;
xy[1] = curxy[1] - un->th * 2 ;
xy[2] = xy[0] + un->tw * ( nele + 1 ) ;
xy[3] = xy[1] + un->th * 5 / 2 ;
setcolor( col.reve ) ;
setlinestyle( SOLID_LINE, 0, NORM_WIDTH ) ;
rectangle( xy[0], xy[1], xy[2], xy[3] ) ;

setcolor( col.tere ) ;
setfillstyle( SOLID_FILL, col.fove ) ;

outtextxy( curxy[0], curxy[1], mens ) ;

ele = 0 ;
flag = 0 ;

do      {
        while( !bioskey( 1 ) ) ;
        c.i = bioskey( 0 ) ;

        switch( c.ch[0] ) {

                case ESC :
                        return( -1 ) ;
                case CR :
```

```
                        if( flag = = 0 )
                                return( 0 ) ;
                        if( strlen( cadena ) = = 5 ) {
                                for( i=0 ; i < 5 ; i++ )
                                        cadaux[i] = tolower( cadena[i] ) ;/*printf("\nCadena[%d] =
%c Cadaux[%d] = %c",i,cadena[i],i,cadaux[i]);}*/

                                cadaux[5] = '\0' ;                      /*printf("\nCadaux =
%s",cadaux);*/
                                if( strcmp( cadaux, "*.ana" ) = = NULL )
                                        return( 0 ) ;
                                }
                        return( 1 ) ;
                case 0 :
                        if( c.ch[1] != 75 )
                                break ;
                case DEL :
                        if( ele != 0 ) {

                                ele-- ;
                                cadena[ele] = '\0' ;
                                setcolor( col.fove ) ;

                                xy[0] = curxy[0] + un->tw * ele ;
                                xy[1] = curxy[1] ;
                                xy[2] = xy[0] ;
                                xy[3] = xy[1] - un->th ;
                                xy[4] = xy[0] + un->tw ;
                                xy[5] = xy[3] ;
                                xy[6] = xy[4] ;
                                xy[7] = xy[1] ;

                                fillpoly( 4, xy ) ;
                                setcolor( col.tere ) ;

                                }
                        else if( flag = = 0 ) {
                                setcolor( col.fove ) ;

                                xy[0] = curxy[0] ;
                                xy[1] = curxy[1] ;
                                xy[2] = xy[0] ;
                                xy[3] = xy[1] - un->th ;
                                xy[4] = xy[0] + un->tw * 5 ;
                                xy[5] = xy[3] ;
                                xy[6] = xy[4] ;
                                xy[7] = xy[1] ;

                                fillpoly( 4, xy ) ;
                                setcolor( col.tere ) ;
                                flag = 1 ;
                                }
                        break ;
                default :
                        if( flag = = 0 ) {
                                setcolor( col.fove ) ;

                                xy[0] = curxy[0] ;
                                xy[1] = curxy[1] ;
                                xy[2] = xy[0] ;
                                xy[3] = xy[1] - un->th ;
```

```
                                        xy[4] = xy[0] + un->tw * 5 ;
                                        xy[5] = xy[3] ;
                                        xy[6] = xy[4] ;
                                        xy[7] = xy[1] ;

                                        fillpoly( 4, xy ) ;
                                        setcolor( col.tere ) ;
                                        flag = 1 ;
                                        }

                                if( ele < nele ) {
                                        cadena[ele] = c.ch[0] ;
                                        outtextxy( curxy[0], curxy[1], cadena ) ;
                                        ele++ ;
                                        }
                        }
                }
while( 1 ) ;

}

/*-------------------------------------------------------------------*/
/*      LEFLO           LECTURA DE REAL(FLOAT) DE VENTANA            */
/*-------------------------------------------------------------------*/

int     leflo( un, col, curxy, real, nent, ndec )

struct unidad   *un     ;
struct color    col     ;
int             curxy[ ];
float           *real   ;
int             nent    ;
int             ndec    ;

{

int             nele    ;
int             ele     ;
int             flag    ;
int             adi     ;
int             xy[8]   ;
char            *dum    ;
char            *mem_char(int,char * );
union inkey     {
                char    ch[2]   ;
                int     i       ;
                }       c       ;


nele = nent + ndec + 1 ;
dum = mem_char( (nele+1), "leflo(dum)" ) ;

xy[0] = curxy[0] - ( un->tw + 1 ) / 2 ;
xy[1] = curxy[1] - un->th * 7 / 4 ;
xy[2] = xy[0] + un->tw * ( nele + 1 ) ;
xy[3] = xy[1] + un->th * 9 / 4 ;
setcolor( col.reve ) ;
setlinestyle( SOLID_LINE, 0, NORM_WIDTH ) ;
rectangle( xy[0], xy[1], xy[2], xy[3] ) ;
```

```
setcolor( col.tere ) ;
setfillstyle( SOLID_FILL, col.fove ) ;

ele = 0 ;
flag = 0 ;
adi = 0 ;

do      {
        while( !bioskey( 1 ) ) ;
        c.i = bioskey( 0 ) ;

        switch( c.ch[0] ) {

                case ESC :
                        free( dum ) ;
                        return( -1 ) ;
                case '.' :
                        if( flag == 1 )
                                break ;
                        flag = 1 ;
                case '0' :
                case '1' :
                case '2' :
                case '3' :
                case '4' :
                case '5' :
                case '6' :
                case '7' :
                case '8' :
                case '9' :
                        if( ele < nele ) {
                                *(dum+ele) = c.ch[0] ;
                                ele++ ;
                                if( ele == nent && flag == 0 ) {
                                        *(dum+ele) = '.' ;
                                        flag = 1 ;
                                        ele++ ;
                                        }
                                outtextxy( curxy[0], curxy[1], dum ) ;
                                }
                        break ;
                case 0 :
                        if( c.ch[1] != 75 )
                                break ;
                case DEL :
                        if( ele != 0 ) {

                                if( (ele-1) == nent && *(dum+ele-1) == '.' ) {
                                        flag = 0 ;
                                        ele -= 2 ;
                                        adi = 1 ;
                                        }
                                else    {
                                        ele-- ;
                                        if( *(dum+ele) == '.' )
                                                flag = 0 ;
                                        }
                                *(dum+ele) = '\0' ;
                                setcolor( col.fove ) ;

                                xy[0] = curxy[0] + un->tw * ele ;
```

```
                                xy[1] = curxy[1] ;
                                xy[2] = xy[0] ;
                                xy[3] = xy[1] - un->th ;
                                xy[4] = xy[0] + un->tw * ( 1 + adi ) ;
                                xy[5] = xy[3] ;
                                xy[6] = xy[4] ;
                                xy[7] = xy[1] ;

                                adi = 0 ;

                                fillpoly( 4, xy ) ;
                                setcolor( col.tere ) ;
                                }
                        break ;
                case CR :
                        *real = atof( dum ) ;
                        free( dum ) ;
                        return( 0 ) ;

                }
        }
while( 1 ) ;

}
```

```
#include "head.h"
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>

extern struct lectur        l       ;
extern struct corfil c      ;
extern struct dibujo        d       ;
extern struct grafic        g       ;

/*-------------------------------------------------------------------*/
/*    LC0                  LECTURA DE CADENA SIMPLE                  */
/*-------------------------------------------------------------------*/

int lc0( fp, s )

FILE            *fp     ;
char            *s      ;

{

int             j       ;


while( fgetc(fp) != ')' ) ;
while( fgetc(fp) != '"' ) ;

for( j=0 ; j<39 ; j++ ) {
        s[j] = fgetc(fp) ;
        if( s[j] == '"' )
                break ;
        }

return( j ) ;

}


/*-------------------------------------------------------------------*/
/*    IG0                  IGUALAR LOS K PRIMEROS CARAC. DE UNA CADENA  */
/*-------------------------------------------------------------------*/

ig0( s0, s1, k )

char            *s0     ;
char            *s1     ;
int             k       ;

{

int             j       ;


for( j=0 ; j<k ; j++ )
        *(s0+j) = *(s1+j) ;

*(s0+k) = NULL ;

}
```

```
/*-------------------------------------------------------------------*/
/*    LIO          LECTURA DE ENTERO                                 */
/*-------------------------------------------------------------------*/

int liO( fp )

FILE            *fp     ;

{

char            s[10]   ;
int             j       ;


while( fgetc(fp) != ')' ) ;

for( j=0 ; j<9 ; j++ ) {
        s[j] = fgetc(fp) ;
        if( s[j] == '\t' || s[j] == '\n' )
                break ;
        }

return( atoi(s) ) ;

}


/*-------------------------------------------------------------------*/
/*    ORIBO              ORDEN RELATIVO DE BLOQUE IF Y BUCLE ( principio ) */
/*-------------------------------------------------------------------*/

int orlBO( bi, bu, pos )

struct bloqif       *bi     ;
struct bucle        *bu     ;
int                 pos     ;

{


if( bu->huev0 > pos || bu->huev1 <= pos )
        return( 0 ) ;
if( bi->hueh1 < bu->hueh1 )
        return( 1 ) ;
if( bi->hueh1 > bu->hueh1 )
        return( 2 ) ;
if( *(bi->huev0) > bu->huev0 )
        return( 1 ) ;
if( *(bi->huev0) < bu->huev0 )
        return( 2 ) ;
if( *(bi->huev1+bi->nmac-1) < bu->huev1 )
        return( 1 ) ;
if( *(bi->huev1+bi->nmac-1) > bu->huev1 )
        return( 2 ) ;
if( bi->nmac > 1 )
        return( 1 ) ;
if( bi->nivani < bu->nivani )
        return( 1 ) ;
else
        return( 2 ) ;
```

```
}

/*-------------------------------------------------------------------------*/
/*     ORIB1                  ORDEN RELATIVO DE BLOQUE IF Y BUCLE ( final )        */
/*-------------------------------------------------------------------------*/

int orIB1( bi, bu, pos )

struct bloqif       *bi     ;
struct bucle        *bu     ;
int                 pos     ;

{

if( bu->huev0 > pos || bu->huev1 <= pos )
        return( 0 ) ;
if( bi->hueh0 > bu->hueh0 )
        return( 1 ) ;
if( bi->hueh0 < bu->hueh0 )
        return( 2 ) ;
if( *(bi->huev0) > bu->huev0 )
        return( 1 ) ;
if( *(bi->huev0) < bu->huev0 )
        return( 2 ) ;
if( *(bi->huev1 + bi->nmac-1) < bu->huev1 )
        return( 1 ) ;
if( *(bi->huev1 + bi->nmac-1) > bu->huev1 )
        return( 2 ) ;
if( bi->nmac > 1 )
        return( 1 ) ;
if( bi->nivani < bu->nivani )
        return( 1 ) ;
else
        return( 2 ) ;

}

/*-------------------------------------------------------------------------*/
/*     ORIB2                  ORDEN RELATIVO DE BLOQUE IF Y BUCLE ( fondo )        */
/*-------------------------------------------------------------------------*/

int orIB2( bi, bu )

struct bloqif       *bi     ;
struct bucle        *bu     ;

{


if( bi->hueh0 > bu->hueh0 )
        return( 1 ) ;
if( bi->hueh0 < bu->hueh0 )
        return( 2 ) ;
if( bi->hueh1 < bu->hueh1 )
        return( 1 ) ;
if( bi->hueh1 > bu->hueh1 )
        return( 2 ) ;
if( *(bi->huev0) > bu->huev0 )
```

```
        return( 1 ) ;
if( *(bi->huev0) < bu->huev0 )
        return( 2 ) ;
if( bi->nmac > 1 )
        return( 1 ) ;
if( bi->nivani < bu->nivani )
        return( 1 ) ;
else
        return( 2 ) ;

}

/*-------------------------------------------------------------------*/
/*    OR0          ORDEN RELATIVO DE DOS ORIGENES                    */
/*-------------------------------------------------------------------*/

int or0( tip1, ori1, tip2, ori2 )

char            tip1    ;
char            *ori1   ;
char            tip2    ;
char            *ori2   ;

{

int             ph0( )  ;
int             pv0( )  ;


if( tip1 == tip2 ) {

        if( strcmp( ori1, ori2 ) == 0 )
                return( 0 ) ;

        if( tip1 == 'o' ) {
                if( strcmp( ori1, "ext" ) == 0 )
                        return( 2 ) ;
                if( strcmp( ori2, "ext" ) == 0 )
                        return( 1 ) ;
                if( strcmp( ori1, "cod" ) == 0 )
                        return( 2 ) ;
                if( strcmp( ori2, "cod" ) == 0 )
                        return( 1 ) ;
                if( pv0( ori1 ) > pv0( ori2 ) )
                        return( 1 ) ;
                else
                        return( 2 ) ;
                }
        }
else    {
        if( tip1 == 'o' )
                return( 1 ) ;
        else
                return( 2 ) ;
        }
}

/*-------------------------------------------------------------------*/
/*    OR1          ORDEN RELATIVO DE DOS DESTINOS                    */
```

```
/*-------------------------------------------------------------------*/

int or1( tip1, des1, tip2, des2 )

char            tip1    ;
char            *des1   ;
char            tip2    ;
char            *des2   ;

{

int             ph0( )  ;
int             pv0( )  ;


if( tip1 == tip2 ) {

        if( strcmp( des1, des2 ) == 0 )
                return( 0 ) ;

        if( tip1 == 'c' ) {
                if( ph0( des1 ) > ph0( des2 ) )
                        return( 1 ) ;
                else
                        return( 2 ) ;
                }
        if( tip1 == 'i' ) {
                if( strcmp( des1, "ext" ) == 0 )
                        return( 1 ) ;
                if( strcmp( des2, "ext" ) == 0 )
                        return( 2 ) ;
                if( pv0( des1 ) < pv0( des2 ) )
                        return( 1 ) ;
                else
                        return( 2 ) ;
                }
        if( tip1 == 'b' ) {
                if( ph0( des1 ) < ph0( des2 ) )
                        return( 1 ) ;
                else
                        return( 2 ) ;
                }
        }
else    {
        if( tip1 == 'c' )
                return( 1 ) ;
        if( tip2 == 'c' )
                return( 2 ) ;
        if( tip1 == 'i' )
                return( 1 ) ;
        if( tip2 == 'i' )
                return( 2 ) ;
        }
}

/*-------------------------------------------------------------------*/
/*      OR2             ORDEN RELATIVO DE DOS INPOUTS                */
/*-------------------------------------------------------------------*/

int or2( or1, or2 )
```

```
struct inpout        *or1    ;
struct inpout        *or2    ;

{


if( &(or1->cv) == &(or2->cv) )
        return( 0 ) ;

if( or1->cv.sr < or2->cv.sr )
        return( 1 ) ;
else if( or1->cv.sr > or2->cv.sr )
        return( 2 ) ;
else    {
        if( or1->cv.th < or2->cv.th )
                return( 1 ) ;
        else if( or1->cv.th > or2->cv.th )
                return( 2 ) ;
        else
                return( 0 ) ;
        }
}

/*-------------------------------------------------------------------------*/
/*      PH0     POSICION HORIZONTAL DE UNA CAJA                            */
/*-------------------------------------------------------------------------*/

int ph0( id )

char                 *id     ;

{

int                  i       ;
struct cajita        *caj    ;


for( i=0 ; i < l.ncajis ; i++ ) {
        caj = (l.cs+i) ;
        if( strcmp( id, caj->id ) == 0 )
                return( caj->pohor ) ;
        }

for( i=0 ; i < l.ncajic ; i++ ) {
        caj = (l.cc+i) ;
        if( strcmp( id, caj->id ) == 0 )
                return( caj->pohor ) ;
        }

return( -1 ) ;

}

/*-------------------------------------------------------------------------*/
/*      PH1             HALLAR POHOR DE UNA CAJA POR SU POVER              */
/*-------------------------------------------------------------------------*/

int ph1( pover )
```

```
int                 pover    ;

{

int                 i        ;


for( i=0 ; i < c.ncajs ; i++ )
        if( pover == (c.cs+i)->pover )
                return( (c.cs+i)->pohor ) ;

for( i=0 ; i < c.ncajc ; i++ )
        if( pover == (c.cc+i)->pover )
                return( (c.cc+i)->pohor ) ;

}

/*-------------------------------------------------------------------*/
/*      IDSEPO          HALLAR ID DE UNA CAJA POR SU POVER           */
/*-------------------------------------------------------------------*/

idsepo( pover, id )

int                 pover    ;
char                id[]     ;

{

int                 i        ;


for( i=0 ; i < l.ncajis ; i++ )
        if( pover == (l.cs+i)->pover ) {
                strcpy( id, (l.cs+i)->id ) ;
                return ;
                }

for( i=0 ; i < l.ncajic ; i++ )
        if( pover == (l.cc+i)->pover ) {
                strcpy( id, (l.cc+i)->id ) ;
                return ;
                }

}


/*-------------------------------------------------------------------*/
/*      PV0     POSICION VERTICAL DE UNA CAJA                        */
/*-------------------------------------------------------------------*/

int pv0( id )

char                *id      ;

{

int                 i        ;
```

```
for( i=0 ; i < l.ncajis ; i++ )
        if( strcmp( id, (l.cs+i)->id ) == 0 )
                return( (l.cs+i)->pover ) ;

for( i=0 ; i < l.ncajic ; i++ )
        if( strcmp( id, (l.cc+i)->id ) == 0 )
                return( (l.cc+i)->pover ) ;

return( -1 ) ;

}

/*-------------------------------------------------------------------*/
/*      ORCAJ           ORDEN DE UNA CAJA SEGUN SU TIPO              */
/*-------------------------------------------------------------------*/

int     orcaj( id, tipo )

char            *id     ;
char            tipo    ;

{

int             i       ;


if( tipo == 's' ) {
        for( i=0 ; i < l.ncajis ; i++ )
                if( strcmp( id, (l.cs+i)->id ) == 0 )
                        return( i ) ;
        }
else    {
        for( i=0 ; i < l.ncajic ; i++ )
                if( strcmp( id, (l.cc+i)->id ) == 0 )
                        return( i ) ;
        }

return( -1 ) ;

}

/*-------------------------------------------------------------------*/
/*   NI0           POSICION DE UN INPUT DE UNA CAJA                 */
/*-------------------------------------------------------------------*/

int ni0( nam, des, tipcaj, orden )

char            *nam    ;
char            *des    ;
char            *tipcaj ;
int             *orden  ;

{

int             i       ;
int             j       ;
struct caja     *caj    ;
```

```
if( strcmp( "ext", des ) == 0 ) {
        *tipcaj = 'e' ;
        *orden = -1 ;
        for( j=0 ; j < c.nout ; j++ )
                if( strcmp( (c.out+j)->s, nam ) == 0 )
                        return( j ) ;
        }

for( i=0 ; i < c.ncajs ; i++ ) {
        caj = (c.cs+i) ;
        if( strcmp( caj->id, des ) == 0 )       {
                *tipcaj = 's' ;
                *orden = i ;
                for( j=0 ; j < caj->ninp ; j++ )
                        if( strcmp( (caj->inp+j)->s, nam ) == 0 )
                                return( j ) ;
                }
        }

for( i=0 ; i < c.ncajc ; i++ ) {
        caj = (c.cc+i) ;
        if( strcmp( caj->id, des ) == 0 )       {
                *tipcaj = 'c' ;
                *orden = i ;
                for( j=0 ; j < caj->ninp ; j++ )
                        if( strcmp( (caj->inp+j)->s, nam ) == 0 )
                                return( j ) ;
                }
        }

return( -1 ) ;

}

/*-------------------------------------------------------------------*/
/*    NI1          POSICION DE UNA ENTRADA DE UNA CAJA               */
/*-------------------------------------------------------------------*/

int ni1( nam, des, tipcaj, orden )

char            *nam    ;
char            *des    ;
char            *tipcaj ;
int             *orden  ;

{

int             i       ;
int             j       ;
struct cajita   *caj    ;


if( strcmp( "ext", des ) == 0 ) {
        *tipcaj = 'e' ;
        *orden = -1 ;
        for( j=0 ; j < l.nsalid ; j++ )
                if( strcmp( (l.sal+j)->nam, nam ) == 0 )
                        return( j ) ;
        }
```

```
for( i=0 ; i < l.ncajis ; i++ ) {
        caj = (l.cs+i) ;
        if( strcmp( caj->id, des ) == 0 )      {
                *tipcaj = 's' ;
                *orden = i ;
                for( j=0 ; j < caj->nentra ; j++ )
                        if( strcmp( (caj->ent+j)->nam, nam ) == 0 )
                                return( j ) ;
                }
        }

for( i=0 ; i < l.ncajic ; i++ ) {
        caj = (l.cc+i) ;
        if( strcmp( caj->id, des ) == 0 )      {
                *tipcaj = 'c' ;
                *orden = i ;
                for( j=0 ; j < caj->nentra ; j++ )
                        if( strcmp( (caj->ent+j)->nam, nam ) == 0 )
                                return( j ) ;
                }
        }

return( -1 ) ;

}


/*-------------------------------------------------------------------------*/
/*     NI2          POSICION DE UNA ENTRADA DE UNA CAJA CONCRETA           */
/*-------------------------------------------------------------------------*/

int ni2( nam, tipcaj, orden )

char            *nam    ;
char            tipcaj  ;
int             orden   ;

{

int             j       ;
struct cajita   *caj    ;


if( tipcaj == 'e' ) {
        for( j=0 ; j < l.nsalid ; j++ )
                if( strcmp( (l.sal+j)->nam, nam ) == 0 )
                        return( j ) ;
        }

else if( tipcaj == 's' ) {
        caj = (l.cs+orden) ;
        for( j=0 ; j < caj->nentra ; j++ )
                if( strcmp( (caj->ent+j)->nam, nam ) == 0 )
                        return( j ) ;
        }

else if( tipcaj == 'c' ) {
        caj = (l.cc+orden) ;
        for( j=0 ; j < caj->nentra ; j++ )
                if( strcmp( (caj->ent+j)->nam, nam ) == 0 )
                        return( j ) ;
```

```
        }
return( -1 ) ;
}

/*-------------------------------------------------------------------*/
/*    NO0          POSICION DE UN OUTPUT DE UNA CAJA                 */
/*-------------------------------------------------------------------*/

int no0( nam, ori, tipcaj, orden )

char            *nam    ;
char            *ori    ;
char            *tipcaj ;
int             *orden  ;

{

int             i       ;
int             j       ;
struct caja     *caj    ;


if( strcmp( "ext", ori ) == 0 ) {
        *tipcaj = 'e' ;
        *orden = -1 ;
        for( j=0 ; j < c.ninp ; j++ )
                if( strcmp( (c.inp+j)->s, nam ) == 0 )
                        return( j ) ;
        }

for( i=0 ; i < c.ncajs ; i++ ) {
        caj = (c.cs+i) ;
        if( strcmp( caj->id, ori ) == 0 )       {
                *tipcaj = 's' ;
                *orden = i ;
                for( j=0 ; j < caj->nout ; j++ )
                        if( strcmp( (caj->out+j)->s, nam ) == 0 )
                                return( j ) ;
                }
        }

for( i=0 ; i < c.ncajc ; i++ ) {
        caj = (c.cc+i) ;
        if( strcmp( caj->id, ori ) == 0 ) {
                *tipcaj = 'c' ;
                *orden = i ;
                for( j=0 ; j < caj->nout ; j++ )
                        if( strcmp( (caj->out+j)->s, nam ) == 0 )
                                return( j ) ;
                }
        }

return( -1 ) ;

}

/*-------------------------------------------------------------------*/
```

```
/*    NO1          POSICION DE UNA SALIDA DE UNA CAJA                    */
/*--------------------------------------------------------------------*/

int no1( nam, ori, tipcaj, orden )

char            *nam    ;
char            *ori    ;
char            *tipcaj ;
int             *orden  ;

{

int             i       ;
int             j       ;
struct cajita   *caj    ;


if( strcmp( "ext", ori ) == 0 ) {
        *tipcaj = 'e' ;
        *orden = -1 ;
        for( j=0 ; j < l.nentra ; j++ )
                if( strcmp( (l.ent+j)->nam, nam ) == 0 )
                        return( j ) ;
        }

for( i=0 ; i < l.ncajis ; i++ ) {
        caj = (l.cs+i) ;
        if( strcmp( caj->id, ori ) == 0 )       {
                *tipcaj = 's' ;
                *orden = i ;
                for( j=0 ; j < caj->nsalid ; j++ )
                        if( strcmp( (caj->sal+j)->nam, nam ) == 0 )
                                return( j ) ;
                }
        }

for( i=0 ; i < l.ncajic ; i++ ) {
        caj = (l.cc+i) ;
        if( strcmp( caj->id, ori ) == 0 )       {
                *tipcaj = 'c' ;
                *orden = i ;
                for( j=0 ; j < caj->nsalid ; j++ )
                        if( strcmp( (caj->sal+j)->nam, nam ) == 0 )
                                return( j ) ;
                }
        }

return( -1 ) ;

}


/*--------------------------------------------------------------------*/
/*    NO2          POSICION DE UNA SALIDA DE UNA CAJA CONCRETA           */
/*--------------------------------------------------------------------*/

int no2( nam, tipcaj, orden )

char            *nam    ;
char            tipcaj  ;
int             orden   ;
```

```
{

int             j       ;
struct cajita   *caj    ;


if( tipcaj == 'e' ) {
        for( j=0 ; j < l.nentra ; j++ )
                if( strcmp( (l.ent+j)->nam, nam ) == 0 )
                        return( j ) ;
        }

else if( tipcaj == 's' ) {
        caj = (l.cs+orden) ;
        for( j=0 ; j < caj->nsalid ; j++ )
                if( strcmp( (caj->sal+j)->nam, nam ) == 0 )
                        return( j ) ;
        }

else if( tipcaj == 'c' ) {
        caj = (l.cc+orden) ;
        for( j=0 ; j < caj->nsalid ; j++ )
                if( strcmp( (caj->sal+j)->nam, nam ) == 0 )
                        return( j ) ;
        }

return( -1 ) ;

}


/*-------------------------------------------------------------------*/
/*    NCL                  TIPO Y ORDEN DE UNA CAJA EN L            */
/*-------------------------------------------------------------------*/

int ncl( iden, tipcaj, orden )

char            *iden   ;
char            *tipcaj ;
int             *orden  ;

{

int             i       ;


if( strcmp( "Exterior", iden ) == 0 ) {
        *tipcaj = 'e' ;
        *orden = -1 ;
        return( 0 ) ;
        }

for( i=0 ; i < l.ncajis ; i++ )
        if( strcmp( (l.cs+i)->id, iden ) == 0 ) {
                *tipcaj = 's' ;
                *orden = i ;
                return( 0 ) ;
                }

for( i=0 ; i < l.ncajic ; i++ )
```

```
        if( strcmp( (l.cc+i)->id, iden ) == 0 )          {
                *tipcaj = 'c' ;
                *orden = i ;
                return( 0 ) ;
                }
return( -1 ) ;

}

/*-------------------------------------------------------------------------*/
/*    HH0BIF                HUECO HORIZONTAL IZQUIERDO DE UN BIF          */
/*-------------------------------------------------------------------------*/

int     hh0bif( i )

int             i       ;

{

int             j       ;
int             hue     ;
struct blif     *bif    ;
int             ph0()   ;


hue = 30000 ;
bif = (l.bif+i) ;

for( j=0 ; j < bif->mac->ncaj ; j++ )
        hue = min( hue, ph0( *(bif->mac->caj+j) ) ) ;

return( hue ) ;

}

/*-------------------------------------------------------------------------*/
/*    SEPINI                NUMERO DE SEPARACIONES EN UN HUECO POR INI    */
/*-------------------------------------------------------------------------*/

int     sepini( hue, pos )

int             hue     ;
int             pos     ;

{

int             i       ;
int             n       ;


n = 0 ;
for( i=0 ; i < c.ninic ; i++ ) {
        if( (c.in+i)->hue == hue && ( (c.in+i)->var->cv.sr / 2 ) == pos )
                n = max( n, strlen( (c.in+i)->valor ) ) ;
        }

return( n ) ;

}
```

```
/*-------------------------------------------------------------------*/
/*    INIL                    INICIALICIACION DE LECTUR              */
/*-------------------------------------------------------------------*/

inil( )

{

char      *mem_char(int,char * );
char      *(*mem_charp(int,char *));


l.nentra = 0 ;
l.nsalid = 0 ;
l.ncajis = 0 ;
l.ncajic = 0 ;
l.nblif = 0 ;
l.nbuc = 0 ;
l.nrecur = 0 ;
l.ninic = 0 ;
l.tree.pohor = 0 ;
l.tree.pover = 0 ;
l.tree.nentra = 0 ;
l.tree.nsalid = 0 ;
l.tree.ncadna = 1 ;

l.id = mem_char( 5, "inil(l.id)" ) ;
l.name = mem_char( 11, "inil(l.name)" ) ;
l.tree.id = mem_char( 5, "inil(l.tree.id)" ) ;
l.tree.name = mem_charp( 1, "inil(l.tree.name)" ) ;
*(l.tree.name) = mem_char( 11, "inil(*l.tree.name)" ) ;

strcpy( l.id, "NADA" ) ;
strcpy( l.name, "SIN NOMBRE" ) ;
strcpy( l.tree.id, "NADA" ) ;
strcpy( *(l.tree.name), "SIN NOMBRE" ) ;

}

/*-------------------------------------------------------------------*/
/*    FREEL       LIBERACION DE MEMORIA DE L                       */
/*-------------------------------------------------------------------*/

freel( )

{

int               i         ;
int               j         ;
int               k         ;


free( l.tree.id ) ;
for( i=0 ; i < l.tree.ncadna ; i++ )
        free( *(l.tree.name+i) ) ;
free( l.tree.name ) ;
if( l.tree.nentra > 0 ) {
        for( i=0 ; i < l.tree.nentra ; i++ ) {
                free( (l.tree.ent+i)->nam ) ;
```

```
                if( (l.tree.ent+i)->norig > 0 ) {
                        for( j=0 ; j < (l.tree.ent+i)->norig ; j++ ) {
                                free( *((l.tree.ent+i)->nom+j) ) ;
                                free( *((l.tree.ent+i)->ori+j) ) ;
                                }
                        free( (l.tree.ent+i)->nom ) ;
                        free( (l.tree.ent+i)->ori ) ;
                        free( (l.tree.ent+i)->tipo ) ;
                        }
                }
        free( l.tree.ent ) ;
        }
if( l.tree.nsalid > 0 ) {
        for( i=0 ; i < l.tree.nsalid ; i++ ) {
                free( (l.tree.sal+i)->nam ) ;
                if( (l.tree.sal+i)->ndest > 0 ) {
                        for( j=0 ; j < (l.tree.sal+i)->ndest ; j++ ) {
                                free( *((l.tree.sal+i)->nom+j) ) ;
                                free( *((l.tree.sal+i)->des+j) ) ;
                                }
                        free( (l.tree.sal+i)->nom ) ;
                        free( (l.tree.sal+i)->des ) ;
                        free( (l.tree.sal+i)->tipo ) ;
                        }
                }
        free( l.tree.sal ) ;
        }
free( l.id ) ;
free( l.name ) ;
if( l.nentra > 0 ) {
        for( i=0 ; i < l.nentra ; i++ ) {
                free( (l.ent+i)->nam ) ;
                if( (l.ent+i)->ndest > 0 ) {
                        for( j=0 ; j < (l.ent+i)->ndest ; j++ ) {
                                free( *((l.ent+i)->nom+j) ) ;
                                free( *((l.ent+i)->des+j) ) ;
                                }
                        free( (l.ent+i)->nom ) ;
                        free( (l.ent+i)->des ) ;
                        free( (l.ent+i)->tipo ) ;
                        }
                }
        free( l.ent ) ;
        }
if( l.nsalid > 0 ) {
        for( i=0 ; i < l.nsalid ; i++ ) {
                free( (l.sal+i)->nam ) ;
                if( (l.sal+i)->norig > 0 ) {
                        for( j=0 ; j < (l.sal+i)->norig ; j++ ) {
                                free( *((l.sal+i)->nom+j) ) ;
                                free( *((l.sal+i)->ori+j) ) ;
                                }
                        free( (l.sal+i)->nom ) ;
                        free( (l.sal+i)->ori ) ;
                        free( (l.sal+i)->tipo ) ;
                        }
                }
        free( l.sal ) ;
        }
if( l.ncajis > 0 ) {
        for( i=0 ; i < l.ncajis ; i++ ) {
```

```
                free( (l.cs+i)->id ) ;
                for( j=0 ; j < (l.cs+i)->ncadna ; j++ )
                        free( *((l.cs+i)->name+j) ) ;
                free( (l.cs+i)->name ) ;
                if( (l.cs+i)->nentra > 0 ) {
                        for( j=0 ; j < (l.cs+i)->nentra ; j++ ) {
                                free( ((l.cs+i)->ent+j)->nam ) ;
                                if( ((l.cs+i)->ent+j)->norig > 0 ) {
                                        for( k=0 ; k < ((l.cs+i)->ent+j)->norig ; k++ ) {
                                                free( *(((l.cs+i)->ent+j)->nom+k) ) ;
                                                free( *(((l.cs+i)->ent+j)->ori+k) ) ;
                                                }
                                        free( ((l.cs+i)->ent+j)->nom ) ;
                                        free( ((l.cs+i)->ent+j)->ori ) ;
                                        free( ((l.cs+i)->ent+j)->tipo ) ;
                                        }
                                }
                        free( (l.cs+i)->ent ) ;
                        }
                if( (l.cs+i)->nsalid > 0 ) {
                        for( j=0 ; j < (l.cs+i)->nsalid ; j++ ) {
                                free( ((l.cs+i)->sal+j)->nam ) ;
                                if( ((l.cs+i)->sal+j)->ndest > 0 ) {
                                        for( k=0 ; k < ((l.cs+i)->sal+j)->ndest ; k++ ) {
                                                free( *(((l.cs+i)->sal+j)->nom+k) ) ;
                                                free( *(((l.cs+i)->sal+j)->des+k) ) ;
                                                }
                                        free( ((l.cs+i)->sal+j)->nom ) ;
                                        free( ((l.cs+i)->sal+j)->des ) ;
                                        free( ((l.cs+i)->sal+j)->tipo ) ;
                                        }
                                }
                        free( (l.cs+i)->sal ) ;
                        }
                }
        free( l.cs ) ;
        }
if( l.ncajic > 0 ) {
        for( i=0 ; i < l.ncajic ; i++ ) {
                free( (l.cc+i)->id ) ;
                for( j=0 ; j < (l.cc+i)->ncadna ; j++ )
                        free( *((l.cc+i)->name+j) ) ;
                free( (l.cc+i)->name ) ;
                if( (l.cc+i)->nentra > 0 ) {
                        for( j=0 ; j < (l.cc+i)->nentra ; j++ ) {
                                free( ((l.cc+i)->ent+j)->nam ) ;
                                if( ((l.cc+i)->ent+j)->norig > 0 ) {
                                        for( k=0 ; k < ((l.cc+i)->ent+j)->norig ; k++ ) {
                                                free( *(((l.cc+i)->ent+j)->nom+k) ) ;
                                                free( *(((l.cc+i)->ent+j)->ori+k) ) ;
                                                }
                                        free( ((l.cc+i)->ent+j)->nom ) ;
                                        free( ((l.cc+i)->ent+j)->ori ) ;
                                        free( ((l.cc+i)->ent+j)->tipo ) ;
                                        }
                                }
                        free( (l.cc+i)->ent ) ;
                        }
                if( (l.cc+i)->nsalid > 0 ) {
                        for( j=0 ; j < (l.cc+i)->nsalid ; j++ ) {
                                free( ((l.cc+i)->sal+j)->nam ) ;
```

```
                                        if( ((l.cc+i)->sal+j)->ndest > 0 ) {
                                                for( k=0 ; k < ((l.cc+i)->sal+j)->ndest ; k++ ) {
                                                        free( *(((l.cc+i)->sal+j)->nom+k) ) ;
                                                        free( *(((l.cc+i)->sal+j)->des+k) ) ;
                                                        }
                                                free( ((l.cc+i)->sal+j)->nom ) ;
                                                free( ((l.cc+i)->sal+j)->des ) ;
                                                free( ((l.cc+i)->sal+j)->tipo ) ;
                                                }
                                        }
                                free( (l.cc+i)->sal ) ;
                                }
                        }
                free( l.cc ) ;
                }
        if( l.nblif > 0 ) {
                for( i=0 ; i < l.nblif ; i++ ) {
                        for( j=0 ; j < (l.bif+i)->nmac ; j++ ) {
                                free( ((l.bif+i)->mac+j)->con ) ;
                                for( k=0 ; k < ((l.bif+i)->mac+j)->ncaj ; k++ )
                                        free( *(((l.bif+i)->mac+j)->caj+k) ) ;
                                free( ((l.bif+i)->mac+j)->caj ) ;
                                }
                        free( (l.bif+i)->mac ) ;
                        if( (l.bif+i)->oc.norig > 0 ) {
                                for( j=0 ; j < (l.bif+i)->oc.norig ; j++ ) {
                                        free( *((l.bif+i)->oc.nom+j) ) ;
                                        free( *((l.bif+i)->oc.ori+j) ) ;
                                        }
                                free( (l.bif+i)->oc.nom ) ;
                                free( (l.bif+i)->oc.ori ) ;
                                free( (l.bif+i)->oc.tipo ) ;
                                }
                        }
                free( l.bif ) ;
                }
        if( l.nbuc > 0 ) {
                for( i=0 ; i < l.nbuc ; i++ ) {
                        for( j=0 ; j < (l.buc+i)->ncaj ; j++ )
                                free( *((l.buc+i)->caj+j) ) ;
                        free( (l.buc+i)->caj ) ;
                        free( (l.buc+i)->texaux ) ;
                        free( (l.buc+i)->texcon ) ;
                        free( (l.buc+i)->excon ) ;
                        if( (l.buc+i)->oc.norig > 0 ) {
                                for( j=0 ; j < (l.buc+i)->oc.norig ; j++ ) {
                                        free( *((l.buc+i)->oc.nom+j) ) ;
                                        free( *((l.buc+i)->oc.ori+j) ) ;
                                        }
                                free( (l.buc+i)->oc.nom ) ;
                                free( (l.buc+i)->oc.ori ) ;
                                free( (l.buc+i)->oc.tipo ) ;
                                }
                        if( (l.buc+i)->ob.norig > 0 ) {
                                for( j=0 ; j < (l.buc+i)->ob.norig ; j++ ) {
                                        free( *((l.buc+i)->ob.nom+j) ) ;
                                        free( *((l.buc+i)->ob.ori+j) ) ;
                                        }
                                free( (l.buc+i)->ob.nom ) ;
                                free( (l.buc+i)->ob.ori ) ;
                                free( (l.buc+i)->ob.tipo ) ;
```

```
                        }
                    if( (l.buc+i)->eb.norig > 0 ) {
                        for( j=0 ; j < (l.buc+i)->eb.norig ; j++ ) {
                            free( *((l.buc+i)->eb.nom+j) ) ;
                            free( *((l.buc+i)->eb.ori+j) ) ;
                            }
                        free( (l.buc+i)->eb.nom ) ;
                        free( (l.buc+i)->eb.ori ) ;
                        free( (l.buc+i)->eb.tipo ) ;
                        }
                    }
            free( l.buc ) ;
            }
    if( l.nrecur > 0 ) {
            for( i=0 ; i < l.nrecur ; i++ ) {
                    free( (l.rec+i)->caja ) ;
                    free( (l.rec+i)->tex ) ;
                    }
            free( l.rec ) ;
            }
    if( l.ninic > 0 ) {
            for( i=0 ; i < l.ninic ; i++ ) {
                    free( (l.ini+i)->caja ) ;
                    free( (l.ini+i)->inp ) ;
                    free( (l.ini+i)->val ) ;
                    free( (l.ini+i)->anca ) ;
                    }
            free( l.ini ) ;
            }

    }

    /*-------------------------------------------------------------------------*/
    /*     FREED                   LIBERACION DE MEMORIA DE D Y RESTOS DE C          */
    /*-------------------------------------------------------------------------*/

    freed( )

    {

    int                 i           ;
    int                 j           ;


                                                                    /*printf("\n Memoria
    antes de freed = %u",coreleft( ) ) ;*/
            for( i=0 ; i < d.re.li.nlinf ; i++ ) {
                    if( (d.re.li.lf+i)->rx0 == 1 )
                            free( (d.re.li.lf+i)->x0 ) ;
                    if( (d.re.li.lf+i)->ry0 == 1 )
                            free( (d.re.li.lf+i)->y0 ) ;
                    if( (d.re.li.lf+i)->rx1 == 1 )
                            free( (d.re.li.lf+i)->x1 ) ;
                    if( (d.re.li.lf+i)->ry1 == 1 )
                            free( (d.re.li.lf+i)->y1 ) ;
                    }                                       /*printf("\n Memoria despues de li. fi. =
    %u",coreleft( ) ) ;*/
            for( i=0 ; i < d.re.nfle ; i++ ) {
                    if( (d.re.fl+i)->rch == 1 )
                            free( (d.re.fl+i)->ch ) ;
```

```
                    if( (d.re.fl+i)->rcv == 1 )
                            free( (d.re.fl+i)->cv ) ;
                    }                                       /*printf("\n Memoria despues de flechas =
%u",coreleft( ) ) ;*/
if( d.re.te.ntcaj > 0 )
            free( d.re.te.tc ) ;                            /*printf("\n Memoria despues de te.tc =
%u",coreleft( ) ) ;*/
if( d.re.te.ntdat > 0 )
            free( d.re.te.td ) ;                            /*printf("\n Memoria despues de te.td =
%u",coreleft( ) ) ;*/
if( d.re.li.nling > 0 )
            free( d.re.li.lg ) ;                            /*printf("\n Memoria despues de li. gr. =
%u",coreleft( ) ) ;*/
if( d.re.li.nlinf > 0 ) {
            free( d.re.li.lf ) ;
            }
if( d.re.nfle > 0 ) {
            free( d.re.fl ) ;
            }                                               /*printf("\n Memoria despues de li. y fl. =
%u",coreleft( ) ) ;*/
if( d.re.narc > 0 )
            free( d.re.ar ) ;                               /*printf("\n Memoria despues de arcos =
%u",coreleft( ) ) ;*/
if( d.re.nrect > 0 )
            free( d.re.re ) ;                               /*printf("\n Memoria despues de rectan. =
%u",coreleft( ) ) ;*/
free( d.dh.nth ) ;
free( d.dh.nvu ) ;
free( d.dh.ntw ) ;
free( d.dh.nhu ) ;
if( d.dc.ncah > 0 )
            free( d.dc.cah ) ;
if( d.dc.ncav > 0 )
            free( d.dc.cav ) ;
free( d.re.te.tt.s ) ;                                      /*printf("\n Memoria despues de d = %u",coreleft( ) )
;*/
if( d.re.nmar1 > 0 )
            free( d.re.ma1 ) ;
if( d.re.nmar2 > 0 )
            free( d.re.ma2 ) ;

if( c.ninic )
            free( c.in ) ;                                  /*printf("\n Memoria despues de c.inic =
%u",coreleft( ) ) ;*/
if( c.nrecur )
            free( c.re ) ;                                  /*printf("\n Memoria despues de c.recur =
%u",coreleft( ) ) ;*/
if( c.nbuc > 0 ) {
            for( i=0 ; i < c.nbuc ; i++ ) {
                    if( (c.bu+i)->nexbu > 0 ) {
                            for( j=0 ; j < (c.bu+i)->nexbu ; j++ )
                                    if( strcmp( (*((c.bu+i)->eb+j))->s, "cod" ) == 0 ) {
                                            free( (*((c.bu+i)->eb+j))->s ) ;
                                            free( *((c.bu+i)->eb+j) ) ;
                                            }
                            free( (c.bu+i)->eb ) ;
                            }
                    if( (c.bu+i)->norbu > 0 ) {
                            for( j=0 ; j < (c.bu+i)->norbu ; j++ )
                                    if( strcmp( (*((c.bu+i)->ob+j))->s, "cod" ) == 0 ) {
                                            free( (*((c.bu+i)->ob+j))->s ) ;
```

```
                                        free( *((c.bu+i)->ob+j) ) ;
                                        }
                                free( (c.bu+i)->ob ) ;
                                }
                        if( (c.bu+i)->norcon > 0 ) {
                                for( j=0 ; j < (c.bu+i)->norcon ; j++ )
                                        if( strcmp( (*((c.bu+i)->oc+j))->s, "cod" ) == 0 ) {
                                                free( (*((c.bu+i)->oc+j))->s ) ;
                                                free( *((c.bu+i)->oc+j) ) ;
                                                }
                                free( (c.bu+i)->oc ) ;
                                }
                        }
                free( c.bu ) ;
                }                                               /*printf("\n Memoria despues de c.bu =
%u",coreleft( ) ) ;*/
if( c.nbloif > 0 ) {
                for( i=0 ; i < c.nbloif ; i++ ) {
                        if( (c.bi+i)->norcon ) {
                                if( strcmp( (*((c.bi+i)->oc))->s, "cod" ) == 0 ) {
                                        free( (*((c.bi+i)->oc))->s ) ;
                                        free( *((c.bi+i)->oc) ) ;
                                        }
                                free( (c.bi+i)->oc ) ;
                                }
                        for( j=0 ; j < (c.bi+i)->nmac ; j++ )
                                free( *((c.bi+i)->con+j) ) ;
                        free( (c.bi+i)->con ) ;
                        free( (c.bi+i)->huev0 ) ;
                        free( (c.bi+i)->huev1 ) ;
                        free( (c.bi+i)->nivcon ) ;
                        free( (c.bi+i)->ncaj ) ;
                        }
                free( c.bi ) ;
                }                                               /*printf("\n Memoria despues de c.bi = %u",coreleft(
) ) ;*/
if( c.nfled ) {
                for( i=0 ; i < c.nfled ; i++ ) {
                        free( (c.fd+i)->de ) ;
                        free( (c.fd+i)->or ) ;
                        }
                free( c.fd ) ;
                }                                               /*printf("\n Memoria despues de c.fd =
%u",coreleft( ) ) ;*/
if( c.ncajs > 0 ) {
                for( i=0 ; i < c.ncajs ; i++ ) {
                        if( (c.cs+i)->ninp > 0 )
                                free( (c.cs+i)->inp ) ;
                        if( (c.cs+i)->nout > 0 )
                                free( (c.cs+i)->out ) ;
                        }
                free( c.cs ) ;
                }
if( c.ncajc > 0 ) {
                for( i=0 ; i < c.ncajc ; i++ ) {
                        if( (c.cc+i)->ninp > 0 )
                                free( (c.cc+i)->inp ) ;
                        if( (c.cc+i)->nout > 0 )
                                free( (c.cc+i)->out ) ;
                        }
                free( c.cc ) ;
```

```
        }                                                   /*printf("\n Memoria despues de cajas =
%u",coreleft( ) ) ;*/
if( c.nout > 0 )
        free( c.out ) ;                                     /*printf("\n Memoria despues de c.out =
%u",coreleft( ) ) ;*/
if( c.ninp > 0 )
        free( c.inp ) ;                                     /*printf("\n Memoria despues de c.inp osea c =
%u",coreleft( ) ) ;*/

}

/*-------------------------------------------------------------------*/
/*    FREEG                LIBERACION DE MEMORIA DE G                */
/*-------------------------------------------------------------------*/

freeg( )

{

int             i       ;


if( g.nrect > 0 )
        free( g.re ) ;
if( g.nline > 0 )
        free( g.li ) ;
if( g.nfle > 0 )
        free( g.fl ) ;
if( g.narc > 0 )
        free( g.ar ) ;
if( g.tx.ntcj > 0 ) {
        for( i=0 ; i < g.tx.ntcj ; i++ )
                free( (g.tx.tgc+i)->s ) ;
        free( g.tx.tgc ) ;
        }
if( g.tx.ntd > 0 ) {
        for( i=0 ; i < g.tx.ntd ; i++ )
                free( (g.tx.tgd+i)->s ) ;
        free( g.tx.tgd ) ;
        }
free( g.tx.tgt.s ) ;
if( g.nmar > 0 ) {
        for( i=0 ; i < g.nmar ; i++ )
                free( (g.ma+i)->tgm.s ) ;
        free( g.ma ) ;
        }

}

/*-------------------------------------------------------------------*/
/*    MERELL                    RELLENAR LO DE DEBAJO               */
/*-------------------------------------------------------------------*/

merell( xy, col )

int             xy[ ]   ;
struct color    col     ;

{
```

```
setcolor( col.fond ) ;
setfillstyle( SOLID_FILL, col.fond ) ;
fillpoly( 4, xy ) ;

}


/*-------------------------------------------------------------------*/
/*    MECOOR                PONER COORDENADAS DEL MENU               */
/*-------------------------------------------------------------------*/

mecoor( un, xymax, xy, k, lt, enme )

struct unidad     *un      ;
int               xymax[]  ;
int               xy[ ]    ;
int               k        ;
int               lt       ;
int               enme     ;

{

int               x        ;
int               y        ;


x = un->tw * ( lt + 4 ) ;
y = ( un->th * ( 5 * k + 4 ) + 1 ) / 2 ;

xy[0] = un->tw * 3 * enme ;
xy[1] = xymax[1] - y - un->th * 2 * enme ;
xy[2] = xy[0] + x ;
xy[3] = xy[1] ;
xy[4] = xy[2] ;
xy[5] = xy[1] + y ;
xy[6] = xy[0] ;
xy[7] = xy[5] ;

}
```

ANEJO II

SELECCIÓN DEL LISTADO DEL PROGRAMA VERSIÓN 2

```
/*-------------------------------------------------------------------*/
/*      DIB.C              FUNCIONES RELATIVAS A LA CREACION DE DIBUJO      */
/*-------------------------------------------------------------------*/

#include <string.h>

#include <diag.var>
#include <dibu.var>
#include <def.var>

#include <dib.fun>
#include <s1.fun>
#include <mem_dibu.fun>
#include <mem.fun>

/*-------------------------------------------------------------------*/
/*      dib_ie             Rellenar una iof con una entsal y poner orden a -1  */
/*-------------------------------------------------------------------*/

void dib_ie( iof_s   *iof,                    /* Estructura iof a rellenar */
             char    *id,                     /* Identificador de la funci¢n tratada */
             entsal_s *ensa )                 /* Estructura entsal que se va a utilizar */
{

iof->fun       = mem_char( (strlen(id) + 1), iof->fun ) ;
iof->fun       = strcpy( iof->fun, id ) ;
iof->orden     = -1 ;
iof->inpout.nom = mem_char( (strlen(ensa->nombre) + 1), iof->inpout.nom ) ;
iof->inpout.nom = strcpy( iof->inpout.nom, ensa->nombre ) ;

}

/*-------------------------------------------------------------------*/
/*      dib21              Rellenar datos generales de inter              */
/*-------------------------------------------------------------------*/

inter_s dib21( lfunc_s      *lfunc )          /* Lista de funciones de las cuales se crea inter */
{
             int  i ;                         /* Contador de bucle */
             int  j ;                         /* Contador de bucle */
             inter_s      inter ;             /* Estructura inter que se va a devolver */

inter.entradas.n = 0 ;
inter.salidas.n  = 0 ;
inter.linterf.n  = 0 ;
inter.entradas.iof   = NULL ;
inter.salidas.iof    = NULL ;
inter.linterf.interf = NULL ;
inter.entradas.m = 0 ;
inter.salidas.m  = 0 ;
inter.linterf.m  = 0 ;

for( i=0 ; i<lfunc->n ; i++ ) {
        inter.entradas = mem_liof( (lfunc->func[i].entradas.n + inter.entradas.n), &(inter.entradas) ) ;
        for ( j=0 ; j<lfunc->func[i].entradas.n ; j++ ) {
                dib_ie( &(inter.entradas.iof[inter.entradas.n]), lfunc->func[i].id, &(lfunc->func[i].entradas.ensa[j]) )
;
                inter.entradas.n++ ;
                }
        }
for( i=0 ; i<lfunc->n ; i++ ) {
```

```
        inter.salidas = mem_liof( (lfunc->func[i].salidas.n + inter.salidas.n), &(inter.salidas) ) ;
        for ( j=0 ; j<lfunc->func[i].salidas.n ; j++ ) {
                dib_ie( &(inter.salidas.iof[inter.salidas.n]), lfunc->func[i].id, &(lfunc->func[i].salidas.ensa[j]) ) ;
                inter.salidas.n++ ;
                }
        }

for( i=0 ; i<lfunc->n ; i++ )
        if( lfunc->func[i].transp == 's' )
                inter.linterf = dib211( &(inter.linterf), &(lfunc->func[i]) ) ;

return( inter ) ;

}

/*-------------------------------------------------------------------*/
/*      dib211          Rellenar otro nivel general de inter         */
/*-------------------------------------------------------------------*/

linterf_s dib211( linterf_s     *linterf,       /* Estructura linterf a rellenar */
                  diag_s  *func )               /* Funci¢n de la cual se crea linterf */
{

*linterf = mem_linterf( (linterf->n + 1), linterf ) ;

dib211a( &(linterf->interf[linterf->n]), func->id ) ;
linterf->interf[linterf->n].inter = dib21( &(func->interior.macro->lfunc) ) ;
linterf->n++ ;

return( *linterf ) ;
}

/*-------------------------------------------------------------------*/
/*      dib211a         Poner nombre en fun de linterf               */
/*-------------------------------------------------------------------*/

void dib211a( interf_s      *interf,        /* Estructura interf a rellenar */
              char *id )                    /* Identificador que tenemos que poner */
{

interf->fun = mem_char( (strlen(id) + 1), interf->fun ) ;
interf->fun = strcpy( interf->fun, id ) ;

}

/*-------------------------------------------------------------------*/
/*      dib224a   Verificar si estan ordenadas las salidas de esa funci¢n */
/*-------------------------------------------------------------------*/

s_n_e dib224a( diag_s *di,                  /* Funci¢n en cuesti¢n */
               liof_s *salidas )            /* Salidas en inter */
{
               int   i ;                    /* Contador de bucle */
               s_n_e  x ;                   /* Respuesta a dar */

if( di->salidas.n == 0 )
        return( SI ) ;

for( i=0 ; i<salidas->n ; i++ )
        if( strcmp( di->id, salidas->iof[i].fun ) == NULL &&
            strcmp( di->salidas.ensa->nombre, salidas->iof[i].inpout.nom ) == NULL ) {
```

```
                if( salidas->iof[i].orden == -1 )
                        x = NO ;
                else
                        x = SI ;
                break ;
                }

return( x ) ;

}

/*-------------------------------------------------------------------*/
/*      dib224c         Comparar dos salidas                         */
/*-------------------------------------------------------------------*/

a_d_e dib224c( entsal_s  *sal1,                 /* Primera salida */
               entsal_s  *sal2,                 /* Segunda salida  */
               char      *id,                   /* Identificador de la caja origen */
               liof_s    *entradas,             /* Lista de las entradas para saber si estan ya ordenadas */
               macro_s   *macro )               /* Macro en la que se esta trabajando */
{
               int       i ;                    /* Contador de bucle */
               int       j ;                    /* Contador de bucle */
               int       pto1 ;                 /* Contador de puntos a favor de la primera salida */
               int       pto2 ;                 /* Contador de puntos a favor de la segunda salida */
               a_d_e     x ;                    /* Enumeraci¢n para decidir entre los destinos */

x = dib224c1( id, sal1, sal2, &(macro->lfunc) ) ;

if( x != IGUAL ) return( x ) ;

pto1 = 0 ;
pto2 = 0 ;

for( i=0 ; i<sal1->orde.n ; i++ )
        for( j=0 ; j<sal2->orde.n ; j++ ) {
                x = dib224c2( &(sal1->orde.od[i]), &(sal2->orde.od[j]), id, sal1->nombre, sal2->nombre,
entradas, macro ) ;
                if( x == ANTES ) pto1++ ;
                else if( x == DESPUES ) pto2++ ;
                }

if( pto1 != pto2 ) {
        if( pto1 > pto2 ) x = ANTES ;
        else              x = DESPUES ;
        }
else    {
        i = strcmp( sal1->nombre, sal2->nombre ) ;
        if( i<0 ) x = ANTES ;
        else      x = DESPUES ;
        }

return( x ) ;

}

/*-------------------------------------------------------------------*/
/*      dib224c1        Casos particulares de salidas                */
/*-------------------------------------------------------------------*/

a_d_e dib224c1( char      *id,                  /* Identificador de la caja origen */
```

```
                entsal_s *sal1,                 /* Primera salida */
                entsal_s *sal2,                 /* Segunda salida */
                lfunc_s  *lfunc )       /* Lista de funciones en donde trabajamos */
{
                int     i ;                     /* Entero auxiliar */
                a_d_e   x ;                     /* Lo que se va a devolver */

if( sal1->orde.n == 1 && strcmp( sal1->orde.od->ordes, "cod" ) == NULL ) {
        if( sal2->orde.n > 1 || strcmp( sal2->orde.od->ordes, "cod" ) != NULL )
                return( ANTES ) ;
        else    {
                i = strcmp( sal1->nombre, sal2->nombre ) ;
                if( i<0 ) return( ANTES ) ;
                else    return( DESPUES ) ;
                }
        }
else if( sal2->orde.n == 1 && strcmp( sal2->orde.od->ordes, "cod" ) == NULL )
        return( DESPUES ) ;

x = IGUAL ;

if( sal1->orde.n == 1 && strcmp( sal1->orde.od->ordes, "ext" ) == NULL ) {
        if( sal2->orde.n > 1 || strcmp( sal2->orde.od->ordes, "ext" ) != NULL )
                x = dib224c11( id, sal1->orde.od->nom, lfunc ) ;
        }
else if( sal2->orde.n == 1 && strcmp( sal2->orde.od->ordes, "ext" ) == NULL )
        x = dib224c11( id, sal2->orde.od->nom, lfunc ) ;

return( x ) ;

}

/*-------------------------------------------------------------------------*/
/*      dib224c11       Verificar si es la £ltima funci¢n                  */
/*-------------------------------------------------------------------------*/

a_d_e dib224c11( char   *id,                    /* Identificador de la caja origen */
                 char   *nom,                   /* Nombre del dato de llegada al exterior */
                 lfunc_s *lfunc )       /* Lista de funciones en donde trabajamos */
{
                int     i ;                     /* Contador de bucle */
                int     fun ;                   /* N§ de la funci¢n buscada */
                int     ent ;                   /* N§ de la entrada buscada */
                int     pover ;         /* Posici¢n vertical de las funciones */
                int     poveraux ;              /* Posici¢n vertical de las funciones auxiliar */

s1_ne( lfunc, "ext", nom, &fun, &ent ) ;

pover = -1 ;
for( i=0 ; i<lfunc->func[fun].entradas.ensa[ent].orde.n ; i++ ) {
        poveraux = s1_pv( lfunc, lfunc->func[fun].entradas.ensa[ent].orde.od[i].ordes ) ;
        pover = ( poveraux > pover ) ? poveraux : pover ;
        }

if( s1_pv( lfunc, id ) == pover )
        return( ANTES ) ;
else
        return( IGUAL ) ;

}
```

```
/*-------------------------------------------------------------------*/
/*      dib224c2        Partido entre dos destinos                   */
/*-------------------------------------------------------------------*/

a_d_e dib224c2( orides_s *od1,                  /* Primer destino */
                orides_s *od2,                  /* Segundo destino */
                char     *id,                   /* Identificador de la caja origen */
                char     *nom1,                 /* Nombre del primer origen */
                char     *nom2,                 /* Nombre del segundo origen */
                liof_s   *entradas,             /* Lista de las entradas para saber si estan ya ordenadas */
                macro_s  *macro )       /* Macro en la que se esta trabajando */
{
                int      phb1 ;                 /* Posici¢n de bajada del primer destino */
                int      phb2 ;                 /* Posici¢n de bajada del segundo destino */
                a_d_e    x ;                    /* Respuesta de las funciones y valor que se devuelve */

phb1 = dib224c21( id, nom1, od1, macro ) ;
phb2 = dib224c21( id, nom2, od2, macro ) ;

if( od1->tipo == od2->tipo ) {
        if( od1->tipo == 'c' )
                x = dib224c22( phb1, phb2 ) ;
        else if( od1->tipo == 'i' )
                x = dib224c23( phb1, phb2, od1, od2, entradas, &(macro->lfunc) ) ;
        else
                x = dib224c24( phb1, phb2, id, nom1, nom2, macro ) ;
        }
else
        x = dib224c25( phb1, phb2, od1->tipo, od2->tipo ) ;

return( x ) ;

}


/*-------------------------------------------------------------------*/
/*      dib224c21       Calcular la bajada del dato                  */
/*-------------------------------------------------------------------*/

int dib224c21( char     *id,                    /* Funci¢n origen del dato */
               char *nom,                       /* Nombre del dato origen */
               orides_s *od,                    /* destino del dato */
               macro_s  *macro )        /* Macro en la que se trabaja */
{

if( od->tipo == 'c' )
        return( s1_pv( &(macro->lfunc), od->ordes ) ) ;
else if( od->tipo == 'i' )
        return( dib224c211( od, &(macro->lfunc) ) ) ;
else
        return( dib224c212( id, nom, macro ) ) ;

}

/*-------------------------------------------------------------------*/
/*      dib224c211      Calcular la bajada del dato de tipo 'i'      */
/*-------------------------------------------------------------------*/

int dib224c211( orides_s *od,                   /* destino del dato */
                lfunc_s  *lfunc )       /* Lista de funciones */
{
```

```
                int     i ;                     /* Contador de bucle */
                int     fun ;                   /* entero que nos indica la posici¢n en la lista de la funci¢n
deseada */
                int     ent ;                   /* entero que nos indica la posici¢n en la funci¢n de la
entrada deseada */
                int     ph ;                    /* Posici¢n horizontal del £ltimo origen */
                int     phaux ;         /* Posici¢n horizontal auxiliar */

s1_ne( lfunc, od->ordes, od->nom, &fun, &ent ) ;

ph = -1 ;
for( i=0 ; i<lfunc->func[fun].entradas.ensa[ent].orde.n ; i++ )
        if( lfunc->func[fun].entradas.ensa[ent].orde.od[i].tipo == 'o' ) {
                phaux = s1_ph( lfunc, lfunc->func[fun].entradas.ensa[ent].orde.od[i].ordes ) ;
                ph = ( phaux > ph ) ? phaux : ph ;
                }

return( ph + 1 ) ;

}

/*-------------------------------------------------------------------*/
/*      dib224c212      Calcular la bajada del dato de tipo 'b'      */
/*-------------------------------------------------------------------*/

int dib224c212( char    *id,                    /* Funci¢n origen */
                char    *nom,                   /* Dato origen */
                macro_s *macro )        /* Macro en que se trabaja */
{
                bucl_s  *bu1 ;                  /* Bucle auxiliar */

bu1 = s1_busbu( id, nom, macro ) ;
return( bu1->pohor1 ) ;

}

/*-------------------------------------------------------------------*/
/*      dib224c22       Decidir entre dos destinos tipo 'c'          */
/*-------------------------------------------------------------------*/

a_d_e dib224c22( int    phb1,                   /* Bajada del primer destino */
                 int    phb2 )                  /* Bajada del segundo destino */
{
                a_d_e   x ;                     /* Enumeraci¢n de respuesta */

if( phb1 == phb2 )
        x = IGUAL ;
else if( phb1 > phb2 )
        x = ANTES ;
else
        x = DESPUES ;

return( x ) ;

}

/*-------------------------------------------------------------------*/
/*      dib224c23       Decidir entre dos destinos tipo 'i'          */
/*-------------------------------------------------------------------*/
```

```
a_d_e dib224c23( int          phb1,                    /* Bajada del primer destino */
                 int          phb2,                    /* Bajada del segundo destino */
                 orides_s *des1,          /* Primer destino */
                 orides_s *des2,          /* Segundo destino */
                 liof_s       *entradas,               /* Entradas en inter para saber si estan ordenadas */
                 lfunc_s  *lfunc )        /* Lista de funciones */
{
                 a_d_e        x ;                      /* Enumeraci¢n de respuesta */
                 int      pover1 ;         /* Posici¢n vertical del primer destino */
                 int          pover2 ;                 /* Posici¢n vertical del segundo destino */

if( phb1 > phb2 )
        x = ANTES ;
else if( phb1 < phb2 )
        x = DESPUES ;
else    {
        pover1 = s1_pv( lfunc, des1->ordes ) ;        /* Calcula la posici¢n vertical del destino */
        pover2 = s1_pv( lfunc, des2->ordes ) ;        /* Idem */
        if( pover1 < pover2 )
                x = ANTES ;
        else if( pover1 > pover2 )
                x = DESPUES ;
        else
                x = dib224c231( des1, des2, entradas ) ;      /* Mira a ver si estan ordenadas y en tal caso
decide */
        }

return( x ) ;

}

/*-------------------------------------------------------------------*/
/*      dib224c231      Comprobar si estan ordenados los destinos y decidir */
/*-------------------------------------------------------------------*/

a_d_e dib224c231( orides_s *des1,                      /* Primer destino */
                  orides_s          *des2,                     /* Segundo destino */
                  liof_s    *entradas )              /* Entradas en inter para saber si estan ordenadas */
{
                  int           i ;                  /* Contador de bucle */
                  int               j ;                      /* Contador de bucle */
                  a_d_e             x ;                      /* Enumeraci¢n de respuesta */

for( i=0 ; i<entradas->n ; i++ )
        if( strcmp( des1->ordes, entradas->iof[i].fun ) == NULL &&
           strcmp( des1->nom, entradas->iof[i].inpout.nom ) == NULL ) {
                if( entradas->iof[i].orden != -1 )
                        for( j=0 ; j<entradas->n ; j++ )
                                if( strcmp( des2->ordes, entradas->iof[j].fun ) == NULL &&
                                   strcmp( des2->nom, entradas->iof[j].inpout.nom ) == NULL ) {
                                        if( entradas->iof[i].orden < entradas->iof[j].orden )
                                                x = ANTES ;
                                        else
                                                x = DESPUES ;
                                        break ;
                                        }
                else
                        x = IGUAL ;
                break ;
                }
```

```
return( x ) ;

}

/*-------------------------------------------------------------------*/
/*      dib224c24       Decidir entre dos destinos tipo 'b'          */
/*-------------------------------------------------------------------*/

a_d_e dib224c24( int       phb1,                  /* Bajada del primer destino */
                 int       phb2,                  /* Bajada del segundo destino */
                 char      *id,                   /* Funci¢n origen del destino */
                 char      *nom1,                 /* Nombre del primer origen */
                 char      *nom2,                 /* Nombre del segundo origen */
                 macro_s *macro )          /* Macro en la que se esta trabajando */
{
                 a_d_e     x ;                    /* Enumeraci¢n de respuesta */
                 bucl_s  *bu1 ;                   /* Puntero bucle auxiliar 1 */
                 bucl_s    *bu2 ;                 /* Puntero bucle auxiliar 2 */

if( phb1 > phb2 )
        x = ANTES ;
else if( phb1 < phb2 )
        x = DESPUES ;
else    {
        bu1 = s1_busbu( id, nom1, macro ) ;    /* Busca el bucle m s interno que tenga ese origen y devuelve su
direcci¢n */
        bu2 = s1_busbu( id, nom2, macro ) ; /* Idem */
        if( bu1 = = bu2 )
                x = IGUAL ;
        else    {
                if( bu1->pohor0 < bu2->pohor0 )
                        x = ANTES ;
                else if( bu1->pohor0 > bu2->pohor0 )
                        x = DESPUES ;
                else    {
                        if( bu1->nivani > bu2->nivani )
                                x = ANTES ;
                        else
                                x = DESPUES ;
                        }
                }
        }

return( x ) ;

}

/*-------------------------------------------------------------------*/
/*      dib224c25       Decidir entre dos destinos de tipo diferente */
/*-------------------------------------------------------------------*/

a_d_e dib224c25( int       phb1,                  /* Bajada del primer destino */
                 int       phb2,                  /* Bajada del segundo destino */
                 char      tipo1,                 /* Tipo del primer destino */
                 char      tipo2 )                /* Tipo del segundo destino */
{
                 a_d_e     x ;                    /* Enumeraci¢n de respuesta */

if( phb1 > phb2 )
        x = ANTES ;
else if( phb1 < phb2 )
```

```
        x = DESPUES ;
else    {
        if( tipo1 == 'c' )
                x = ANTES ;
        else if( tipo2 == 'c' )
                x = DESPUES ;
        else if( tipo1 == 'i' )
                x = ANTES ;
        else
                x = DESPUES ;
        }

return( x ) ;

}
```

```
/*-------------------------------------------------------------------*/
/* DIB.FUN    PROTOTIPOS DE FUNCIONES RELATIVAS A LA CREACION DE DIBUJO  */
/*-------------------------------------------------------------------*/


/*-------------------------------------------------------------------*/

/*        dib_ie              Rellenar una iof con una entsal y poner orden a -1  */

void dib_ie( iof_s   *,                     /* Estructura iof a rellenar */
             char    *,                     /* Identificador de la funci¢n tratada */
             entsal_s      * ) ;                 /* Estructura entsal que se va a utilizar */

/*-------------------------------------------------------------------*/

/*        dib21               Rellenar datos generales de inter                 */

inter_s dib21( lfunc_s       * ) ;                  /* Lista de funciones de las cuales se crea inter */

/*-------------------------------------------------------------------*/

/*        dib211              Rellenar otro nivel general de inter              */

linterf_s dib211( linterf_s    *,           /* Estructura linterf a rellenar */
                  diag_s   * ) ;            /* Funci¢n de la cual se crea linterf */

/*-------------------------------------------------------------------*/

/*        dib211a             Poner nombre en fun de linterf                    */

void dib211a( interf_s       *,                     /* Estructura interf a rellenar */
              char  * ) ;                   /* Identificador que tenemos que poner */

/*-------------------------------------------------------------------*/

/*        dib224a    Verificar si estan ordenadas las salidas de esa funci¢n  */

s_n_e dib224a( diag_s *,                    /* Funci¢n en cuesti¢n */
               liof_s * ) ;                 /* Salidas en inter */

/*-------------------------------------------------------------------*/

/*        dib224c             Comparar dos salidas                              */

a_d_e dib224c( entsal_s  *,                 /* Primera salida */
               entsal_s  *,                 /* Segunda salida  */
               char      *,                 /* Identificador de la caja origen */
               liof_s    *,                 /* Lista de las entradas para saber si estan ya ordenadas */
               macro_s   * ) ;                  /* Macro en la que se esta trabajando */

/*-------------------------------------------------------------------*/

/*        dib224c1            Casos particulares de salidas                     */

a_d_e dib224c1( char         *,                 /* Identificador de la caja origen */
                entsal_s *,                 /* Primera salida */
                entsal_s *,                 /* Segunda salida */
                lfunc_s  * ) ;                  /* Lista de funciones en donde trabajamos */

/*-------------------------------------------------------------------*/
```

```
/*      dib224c11       Verificar si es la £ltima funci¢n                    */

a_d_e dib224c11( char       *,                  /* Identificador de la caja origen */
                 char       *,                  /* Nombre del dato de llegada al exterior */
                 lfunc_s * ) ;                  /* Lista de funciones en donde trabajamos */

/*-------------------------------------------------------------------------*/

/*      dib224c2        Partido entre dos destinos                           */

a_d_e dib224c2( orides_s *,             /* Primer destino */
                orides_s *,                     /* Segundo destino */
                char       *,                   /* Identificador de la caja origen */
                char       *,                   /* Nombre del primer origen */
                char       *,                   /* Nombre del segundo origen */
                liof_s     *,                   /* Lista de las entradas para saber si estan ya
ordenadas */
                macro_s  * ) ;                  /* Macro en la que se esta trabajando */

/*-------------------------------------------------------------------------*/

/*      dib224c21       Calcular la bajada del dato                          */

int dib224c21( char         *,                  /* Funci¢n origen del dato */
               char *,                  /* Nombre del dato origen */
               orides_s     *,                  /* destino del dato */
               macro_s      * ) ;               /* Macro en la que se trabaja */

/*-------------------------------------------------------------------------*/

/*      dib224c211      Calcular la bajada del dato de tipo 'i'              */

int dib224c211( orides_s *,             /* destino del dato */
                lfunc_s    * ) ;                /* Lista de funciones */

/*-------------------------------------------------------------------------*/

/*      dib224c212      Calcular la bajada del dato de tipo 'b'              */

int dib224c212( char        *,                  /* Funci¢n origen */
                char        *,                  /* Dato origen */
                macro_s   * ) ;                 /* Macro en que se trabaja */

/*-------------------------------------------------------------------------*/

/*      dib224c22       Decidir entre dos destinos tipo'c'                   */

a_d_e dib224c22( int,                           /* Bajada del primer destino */
                 int ) ;                /* Bajada del segundo destino */

/*-------------------------------------------------------------------------*/

/*      dib224c23       Decidir entre dos destinos tipo 'i'                  */

a_d_e dib224c23( int          ,                 /* Bajada del primer destino */
                 int          ,                 /* Bajada del segundo destino */
                 orides_s *,                    /* Primer destino */
                 orides_s *,                    /* Segundo destino */
                 liof_s       *,                /* Entradas en inter para saber si estan ordenadas */
                 lfunc_s  * ) ;                 /* Lista de funciones */
```

```
/*-------------------------------------------------------------------*/

/*      dib224c231      Comprobar si estan ordenados los destinos y decidir */

a_d_e dib224c231( orides_s *,                   /* Primer destino */
                  orides_s      *,                      /* Segundo destino */
                  liof_s    * ) ;               /* Entradas en inter para saber si estan ordenadas */

/*-------------------------------------------------------------------*/

/*      dib224c24       Decidir entre dos destinos tipo 'b'             */

a_d_e dib224c24( int        ,                   /* Bajada del primer destino */
                 int        ,                   /* Bajada del segundo destino */
                 char       *,                  /* Funci¢n origen del destino */
                 char       *,                  /* Nombre del primer origen */
                 char       *,                  /* Nombre del segundo origen */
                 macro_s * ) ;                  /* Macro en la que se esta trabajando */

/*-------------------------------------------------------------------*/

/*      dib224c25       Decidir entre dos destinos de tipo diferente    */

a_d_e dib224c25( int        ,                   /* Bajada del primer destino */
                 int        ,                   /* Bajada del segundo destino */
                 char       ,                   /* Tipo del primer destino */
                 char       ) ;                 /* Tipo del segundo destino */

/*-------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------------*/
/*      ESC.C           Funciones de escribir un fichero                   */
/*-------------------------------------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include <dir.h>
#include <dos.h>

#include <def.var>
#include <diag.var>

#include <esc.fun>

/*-------------------------------------------------------------------------*/
/*      ESC             Escribir un fichero *.ANA                          */
/*-------------------------------------------------------------------------*/

void esc ( diag_s   *di )               /* estructura diagrama a escribir */
{
        FILE            *fp ;           /* puntero fichero */
        char            dum[13] ;       /* cadena auxiliar de fichero *.ana */

esc0( di ) ;

strcpy( dum, di->id ) ;
strcat( dum, ".ana" ) ;

if( ( fp = fopen( dum, "w" ) ) == NULL ) {
        /*closegraph( ) ;*/
        printf("\n Fallo al abrir el fichero %s", dum ) ;
        exit( 1 ) ;
        }

fprintf( fp, "- CABECERA - " ) ;
esc1( fp, di ) ;

fclose( fp ) ;

}

/*-------------------------------------------------------------------------*/
/*      ESC0            Verificar si existe y renombrar el fichero *.ANA   */
/*-------------------------------------------------------------------------*/

void esc0 ( diag_s   *di )              /* estructura diagrama a escribir */
{
        char    dum[13] ;       /* cadena auxiliar de fichero *.ana */
        char    dub[13] ;       /* cadena auxiliar de fichero *.bk  */
        struct ffblk dir ;      /*estructura para funci¢n 'findfirst'*/
        int             at=FA_ARCH ;  /* entero para funci¢n 'findfirst' */

strcpy( dum, di->id ) ;
strcat( dum, ".ana" ) ;

strcpy( dub, di->id ) ;
strcat( dub, ".bk" ) ;

if( !findfirst( dum, &dir, at ) ) {
        if( !findfirst( dub, &dir, at ) )
                if( remove( dub ) == -1 ) {
```

```
                        /*closegraph( ) ;*/
                        printf("\nFallo al borrar el fichero %s", dub ) ;
                        exit( 1 ) ;
                        }
        if( rename( dum, dub ) == -1 ) {
                /*closegraph( ) ;*/
                printf("\nFallo al renombrar el fichero %s en %s", dum, dub ) ;
                exit( 1 ) ;
                }
        }

}

/*-------------------------------------------------------------------------*/
/*      ESC1            Escribir una estructura diagrama                   */
/*-------------------------------------------------------------------------*/

void esc1 ( FILE    *fp,                /* puntero fichero */
            diag_s *di )                /* estructura diagrama a escribir */
{

fprintf( fp, "\n( Identificador ) %c%s%c", '\0', di->id, '\0' ) ;

fprintf( fp, "\n- NOMBRE - " ) ;
esc_lc( fp, &(di->nombre) ) ;

fprintf( fp, "\n( Tipo )  %c%c%c", '\0', di->tipo, '\0' ) ;

fprintf( fp, "\t( Integrado ) %c%c%c", '\0', di->integ, '\0' ) ;

fprintf( fp, "\t( Transparente ) %c%c%c", '\0', di->transp, '\0' ) ;

fprintf( fp, "\t( Posici¢n horizontal ) %c%i%c", '\0', di->pohor, '\0' ) ;

fprintf( fp, "\t( Posici¢n vertical ) %c%i%c", '\0', di->pover, '\0' ) ;

fprintf( fp, "\n- ENTRADAS - " ) ;
esc_les( fp, &(di->entradas) ) ;

fprintf( fp, "\n- SALIDAS - " ) ;
esc_les( fp, &(di->salidas) ) ;

fprintf( fp, "\n- INTERIOR - " ) ;
esc_lm( fp, &(di->interior) ) ;

}

/*-------------------------------------------------------------------------*/
/*      ESC_LB          Escribir una lista de bucles                       */
/*-------------------------------------------------------------------------*/

void esc_lb ( FILE   *fp,               /* puntero fichero */
              lbucl_s       *lis )              /* lista de bucles a escribir*/
{
              int   i ;                 /* contador de bucle */

fprintf( fp, "\n( N§ de bucles ) %c%i%c", '\0', lis->n, '\0' ) ;

for( i=0 ; i<lis->n ; i++ ) {
        fprintf( fp, "\nBUCLE N§%i", i ) ;
        fprintf( fp, "\n( pohor0 )  %c%i%c", '\0', lis->bucl[i].pohor0, '\0' ) ;
```

```
        fprintf( fp, "\t( pohor1 )  %c%i%c", '\0', lis->bucl[i].pohor1, '\0' ) ;
        fprintf( fp, "\t( texaux )  %c%s%c", '\0', lis->bucl[i].texaux, '\0' ) ;
        fprintf( fp, "\t( nivani )  %c%i%c", '\0', lis->bucl[i].nivani, '\0' ) ;
        fprintf( fp, "\t( texcon )  %c%s%c", '\0', lis->bucl[i].texcon, '\0' ) ;
        fprintf( fp, "\t( tipo )  %c%c%c", '\0', lis->bucl[i].tipo, '\0' ) ;
        fprintf( fp, "\t( excon )  %c%s%c", '\0', lis->bucl[i].excon, '\0' ) ;
        fprintf( fp, "\n- ORIGENES DE CONDICION DEL BUCLE -" ) ;
        esc_lod( fp, &(lis->bucl[i].oc) ) ;
        fprintf( fp, "\n- ORIGENES DEL BUCLE -" ) ;
        esc_lod( fp, &(lis->bucl[i].ob) ) ;
        fprintf( fp, "\n- DESTINOS DEL BUCLE -" ) ;
        esc_lod( fp, &(lis->bucl[i].eb) ) ;
        }

}

/*-------------------------------------------------------------------------*/
/*      ESC_LBI         Escribir una lista de bloques if                   */
/*-------------------------------------------------------------------------*/

void esc_lbi ( FILE  *fp,               /* puntero fichero */
               lblif_s      *lis )      /* lista de bloques if a escribir*/
{
               int  i ;                 /* contador de bucle */

fprintf( fp, "\n( N§ de bloques if ) %c%i%c", '\0', lis->n, '\0' ) ;

for( i=0 ; i<lis->n ; i++ ) {
        fprintf( fp, "\nBLOQUE IF N§%i", i ) ;
        fprintf( fp, "\n- BLOQUES DEL BLOQUE IF -" ) ;
        esc_lbl( fp, &(lis->blif[i].lbloc) ) ;
        fprintf( fp, "\n- ORIGENES DE CONDICION DEL BLOQUE IF -" ) ;
        esc_lod( fp, &(lis->blif[i].origenes) ) ;
        }

}

/*-------------------------------------------------------------------------*/
/*      ESC_LBL         Escribir una lista de bloques                      */
/*-------------------------------------------------------------------------*/

void esc_lbl ( FILE  *fp,               /* puntero fichero */
               lbloc_s      *lis )      /* lista de bloques a escribir*/
{
               int  i ;                 /* contador de bucle */

fprintf( fp, "\n( N§ de bloques ) %c%i%c", '\0', lis->n, '\0' ) ;

for( i=0 ; i<lis->n ; i++ ) {
        fprintf( fp, "\n( funci¢n )  %c%s%c", '\0', lis->bloc[i].fun, '\0' ) ;
        fprintf( fp, "\t( condici¢n )  %c%s%c", '\0', lis->bloc[i].con, '\0' ) ;
        }

}

/*-------------------------------------------------------------------------*/
/*      ESC_LC          Escribir una lista de cadenas                      */
/*-------------------------------------------------------------------------*/

void esc_lc ( FILE  *fp,                /* puntero fichero */
              lcad_s        *lis )      /* lista de cadenas a escribir */
```

```
{
        int   i ;                       /* contador de bucle */

fprintf( fp, "\n( N§ de cadenas ) %c%i%c", '\0', lis->n, '\0' ) ;

for( i=0 ; i<lis->n ; i++ )
        fprintf( fp, "\t( %c )  %c%s%c", '0'+i, '\0', lis->cad[i], '\0' ) ;

}

/*-------------------------------------------------------------------*/
/*      ESC_LES         Escribir una lista de entradas/salidas       */
/*-------------------------------------------------------------------*/

void esc_les ( FILE  *fp,               /* puntero fichero */
        lentsal_s *lis )                /* lista de entrada/salida a escribir*/
{
        int   i ;                       /* contador de bucle */

fprintf( fp, "\n( N§ de entsal ) %c%i%c", '\0', lis->n, '\0' ) ;

for( i=0 ; i<lis->n ; i++ ) {
        fprintf( fp, "\nENTSAL N§%i", i ) ;
        fprintf( fp, "\n( Nombre )  %c%s%c", '\0', lis->ensa[i].nombre, '\0' ) ;
        esc_lod( fp, &(lis->ensa[i].orde) ) ;
        }

}

/*-------------------------------------------------------------------*/
/*      ESC_LF          Escribir una lista de funciones              */
/*-------------------------------------------------------------------*/

void esc_lf ( FILE   *fp,               /* puntero fichero */
        lfunc_s         *lis )          /* lista de funciones a escribir */
{
        int   i ;                       /* contador de bucle */

fprintf( fp, "\n( N§ de funciones ) %c%i%c", '\0', lis->n, '\0' ) ;

for( i=0 ; i<lis->n ; i++ ) {
        fprintf( fp, "\nFUNCION N§%i", i ) ;
        esc1( fp, &(lis->func[i]) ) ;
        }

}

/*-------------------------------------------------------------------*/
/*      ESC_LIN         Escribir una lista de inicializaciones       */
/*-------------------------------------------------------------------*/

void esc_lin ( FILE  *fp,               /* puntero fichero */
        linic_s         *lis )          /* lista de recursiones a escribir */
{
        int   i ;                       /* contador de bucle */

fprintf( fp, "\n( N§ de inicializaciones ) %c%i%c", '\0', lis->n, '\0' ) ;

for( i=0 ; i<lis->n ; i++ ) {
        fprintf( fp, "\nINICIALIZACION N§%i", i ) ;
        fprintf( fp, "\n( funci¢n )  %c%s%c", '\0', lis->inic[i].fun, '\0' ) ;
```

```
        fprintf( fp, "\t( entrada )  %c%s%c", '\0', lis->inic[i].inp, '\0' ) ;
        fprintf( fp, "\t( valor )  %c%s%c", '\0', lis->inic[i].val, '\0' ) ;
        fprintf( fp, "\t( pohor )  %c%i%c", '\0', lis->inic[i].pohor, '\0' ) ;
        }

}

/*-------------------------------------------------------------------------*/
/*      ESC_LM          Escribir una lista de macros                       */
/*-------------------------------------------------------------------------*/

void esc_lm ( FILE *fp,                 /* puntero fichero */
              lmacro_s      *lis )      /* lista de macros a escribir*/
{
              int   i ;                 /* contador de bucle */

fprintf( fp, "\n( N§ de macros ) %c%i%c", '\0', lis->n, '\0' ) ;

for( i=0 ; i<lis->n ; i++ ) {
        fprintf( fp, "\n-- MACRO N§%i --", i ) ;
        esc_lf( fp, &(lis->macro[i].lfunc) ) ;
        esc_lbi( fp, &(lis->macro[i].lblif) ) ;
        esc_lb( fp, &(lis->macro[i].lbucl) ) ;
        esc_lr( fp, &(lis->macro[i].lrecu) ) ;
        esc_lin( fp, &(lis->macro[i].linic) ) ;
        }

}

/*-------------------------------------------------------------------------*/
/*      ESC_LOD                 Escribir una lista de origenes/destinos    */
/*-------------------------------------------------------------------------*/

void esc_lod ( FILE *fp,                /* puntero fichero */
               lorides_s *lis )         /* lista de origen/destino a escribir*/
{
               int  i ;                 /* contador de bucle */

fprintf( fp, "\n( N§ de ori/des ) %c%i%c", '\0', lis->n, '\0' ) ;

for( i=0 ; i<lis->n ; i++ ) {
        fprintf( fp, "\n( Nom )  %c%s%c", '\0', lis->od[i].nom, '\0' ) ;
        fprintf( fp, "\t( ordes )  %c%s%c", '\0', lis->od[i].ordes, '\0' ) ;
        fprintf( fp, "\t( tipo )  %c%c%c", '\0', lis->od[i].tipo, '\0' ) ;
        }

}

/*-------------------------------------------------------------------------*/
/*      ESC_LR          Escribir una lista de recursiones                  */
/*-------------------------------------------------------------------------*/

void esc_lr ( FILE   *fp,               /* puntero fichero */
              lrecu_s       *lis )      /* lista de recursiones a escribir */
{
              int   i ;                 /* contador de bucle */

fprintf( fp, "\n( N§ de recursiones ) %c%i%c", '\0', lis->n, '\0' ) ;

for( i=0 ; i<lis->n ; i++ ) {
        fprintf( fp, "\nRECURSION N§%i", i ) ;
```

```
        fprintf( fp, "\n( funci¢n )  %c%s%c", '\0', lis->recu[i].fun, '\0' ) ;
        fprintf( fp, "\t( texto )  %c%s%c", '\0', lis->recu[i].texto, '\0' ) ;
        }

}
```

```
/*-------------------------------------------------------------------*/
/*      ESC.FUN         Funciones de escribir un fichero             */
/*-------------------------------------------------------------------*/

/*      ESC             Escribir un fichero *.ANA                    */

void esc ( diag_s   * ) ;           /* estructura diagrama a escribir */

/*-------------------------------------------------------------------*/

/*      ESC0            Verificar si existe y renombrar el fichero *.ANA */

void esc0 ( diag_s  * ) ;           /* estructura diagrama a escribir */

/*-------------------------------------------------------------------*/

/*      ESC1            Escribir una estructura diagrama             */

void esc1 ( FILE    *,              /* puntero fichero */
            diag_s * ) ;            /* estructura diagrama a escribir */

/*-------------------------------------------------------------------*/

/*      ESC_LB          Escribir una lista de bucles                 */

void esc_lb ( FILE  *,              /* puntero fichero */
              lbucl_s       * ) ;       /* lista de bucles a escribir*/

/*-------------------------------------------------------------------*/

/*      ESC_LBI         Escribir una lista de bloques if             */

void esc_lbi ( FILE *,              /* puntero fichero */
               lblif_s      * ) ;       /* lista de bloques a escribir*/

/*-------------------------------------------------------------------*/

/*      ESC_LBL         Escribir una lista de bloques                */

void esc_lbl ( FILE *,              /* puntero fichero */
               lbloc_s      * ) ;       /* lista de bloques a escribir*/

/*-------------------------------------------------------------------*/

/*      ESC_LC          Escribir una lista de cadenas                */

void esc_lc ( FILE  *,              /* puntero fichero */
              lcad_s        * ) ;       /* lista de cadenas a escribir */

/*-------------------------------------------------------------------*/

/*      ESC_LES         Escribir una lista de entradas/salidas       */

void esc_les ( FILE *,              /* puntero fichero */
               lentsal_s * ) ;      /* lista de entrada/salida a escribir*/

/*-------------------------------------------------------------------*/

/*      ESC_LF          Escribir una lista de funciones              */

void esc_lf ( FILE  *,              /* puntero fichero */
```

```
                lfunc_s         * ) ;               /* lista de funciones a escribir */

/*-------------------------------------------------------------------------*/

/*      ESC_LIN         Escribir una lista de inicializaciones                  */

void esc_lin ( FILE  *,                 /* puntero fichero */
                linic_s         * ) ;           /* lista de recursiones a escribir */

/*-------------------------------------------------------------------------*/

/*      ESC_LM          Escribir una lista de macros                            */

void esc_lm ( FILE  *,                  /* puntero fichero */
                lmacro_s        * ) ;           /* lista de macros a escribir*/

/*-------------------------------------------------------------------------*/

/*      ESC_LOD                 Escribir una lista de origenes/destinos         */

void esc_lod ( FILE  *,                 /* puntero fichero */
                lorides_s * ) ;         /* lista de origen/destino a escribir*/

/*-------------------------------------------------------------------------*/

/*      ESC_LR          Escribir una lista de recursiones                       */

void esc_lr ( FILE   *,                 /* puntero fichero */
                lrecu_s         * ) ;           /* lista de recursiones a escribir */

/*-------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------*/
/*      LEE.C              Subrutinas de leer un fichero                */
/*-------------------------------------------------------------------*/

#include <stdio.h>

#include <def.var>
#include <diag.var>

#include <mem.fun>
#include <mem_diag.fun>
#include <lee.fun>

/*-------------------------------------------------------------------*/
/*      LEE                Leer un fichero *.ANA y dar memoria          */
/*-------------------------------------------------------------------*/

diag_s lee ( char   *dum )          /* Identificador del fichero a leer */
{
        diag_s      di ;            /* estructura diagrama a rellenar */
        FILE  *fp ;                 /* puntero fichero */

if( ( fp = fopen( dum, "r" ) ) == NULL ) {
        /*closegraph( ) ;*/
        printf("\n Fallo al abrir el fichero %s", dum ) ;
        exit( 1 ) ;
        }

di = lee1( fp ) ;

fclose( fp ) ;

return( di ) ;

}

/*-------------------------------------------------------------------*/
/*      LEE1               Leer una estructura diagrama y dar memoria   */
/*-------------------------------------------------------------------*/

diag_s lee1 ( FILE  *fp )           /* Fichero del que se va a leer */
{
        diag_s      di ;            /* estructura diagrama que se va */
                                    /* a rellenar */

di.id       = lee_c( fp ) ;
di.nombre   = lee_lc( fp ) ;
di.tipo     = lee_uc( fp ) ;
di.integ    = lee_uc( fp ) ;
di.transp   = lee_uc( fp ) ;
di.pohor    = lee_i( fp ) ;
di.pover    = lee_i( fp ) ;
di.entradas = lee_les( fp ) ;
di.salidas  = lee_les( fp ) ;
di.interior = lee_lm( fp ) ;

return( di ) ;

}

/*-------------------------------------------------------------------*/
/*      LEE_B              Leer una estructura bucle y dar memoria      */
```

```
/*-------------------------------------------------------------------*/

bucl_s lee_b ( FILE *fp )              /* Fichero del que se va a leer */
{
        bucl_s      buc;               /* estructura bucle que se */
                                       /* va a rellenar */

buc.pohor0 = lee_i( fp ) ;
buc.pohor1 = lee_i( fp ) ;
buc.texaux = lee_c( fp ) ;
buc.nivani = lee_i( fp ) ;
buc.texcon = lee_c( fp ) ;
buc.tipo   = lee_uc( fp ) ;
buc.excon  = lee_c( fp ) ;
buc.oc     = lee_lod( fp ) ;
buc.ob     = lee_lod( fp ) ;
buc.eb     = lee_lod( fp ) ;

return( buc ) ;

}

/*-------------------------------------------------------------------*/
/*      LEE_BI          Leer una estructura bloque if y dar memoria      */
/*-------------------------------------------------------------------*/

blif_s lee_bi ( FILE  *fp )            /* Fichero del que se va a leer */
{
        blif_s      bli;               /* estructura bloque if que se */
                                       /* va a rellenar */

bli.lbloc  = lee_lbl( fp ) ;
bli.origenes = lee_lod( fp ) ;

return( bli ) ;

}

/*-------------------------------------------------------------------*/
/*      LEE_BL          Leer una estructura bloque y dar memoria      */
/*-------------------------------------------------------------------*/

bloc_s lee_bl ( FILE *fp )             /* Fichero del que se va a leer */
{
        bloc_s      bloc;              /* estructura bucle que se */
                                       /* va a rellenar */

bloc.fun = lee_c( fp ) ;
bloc.con = lee_c( fp ) ;

return( bloc ) ;

}

/*-------------------------------------------------------------------*/
/*      LEE_C           Leer una cadena y reservarle memoria          */
/*-------------------------------------------------------------------*/

char *lee_c ( FILE  *fp )              /* Fichero del que se va a leer */
{
        char  *dum ;                   /* cadena intermedia para almacenar */
```

```
                                        /* y reservar memoria */
        int   i ;                       /* contador de bucle */

dum = NULL ;

while( fgetc(fp) != ')' ) ;             /* la cadena debe ir precedida por: */
while( fgetc(fp) != '\0' ) ;            /*              ) \0            */

i = 0 ;
do      {
        i++ ;
        dum = mem_char( i, dum ) ;
        dum[i-1] = fgetc(fp) ;
        if( dum[i-1] == '\0' )                  /* la cadena debe acabar por: \0  */
                /* o retorno de carro*/
                break ;
        }
while( 1 ) ;

return( dum ) ;

}

/*-------------------------------------------------------------------------*/
/*      LEE_ES          Leer una estructura entrada/salida y dar memoria   */
/*-------------------------------------------------------------------------*/

entsal_s lee_es ( FILE          *fp )   /* Fichero del que se va a leer */
{
                entsal_s ensa ;         /* estructura entrada/salida que se */
                                        /* va a rellenar */

ensa.nombre = NULL ;

ensa.nombre = lee_c( fp ) ;
ensa.orde   = lee_lod( fp ) ;

return( ensa ) ;

}

/*-------------------------------------------------------------------------*/
/*      LEE_I           Leer un entero                                     */
/*-------------------------------------------------------------------------*/

int lee_i ( FILE      *fp )             /* Fichero del que se va a leer */

{
        char   dum[10] ;                /* cadena intermedia para almacenar */
                                        /* el entero que devolver  */
        int             i ;             /* contador de bucle */

while( fgetc(fp) != ')' ) ;             /* el entero debe ir precedido por: */
while( fgetc(fp) != '\0' ) ;            /*          ) \0                    */

for( i=0 ; i<9 ; i++ ) {
        dum[i] = fgetc(fp) ;
        if( dum[i] == '\0' )                    /* El entero debe acabar por: \0  */
                break ;
        }
```

```
return( atoi(dum) ) ;

}

/*-------------------------------------------------------------------------*/
/*        LEE_IN              Leer una estructura inicializaci¢n y dar memoria    */
/*-------------------------------------------------------------------------*/

inic_s lee_in ( FILE *fp )              /* Fichero del que se va a leer */
{
                inic_s    in ;              /* estructura inicializaci¢n que se */
                                            /* va a rellenar */

in.fun = NULL ;
in.inp = NULL ;
in.val = NULL ;

in.fun   = lee_c( fp ) ;
in.inp   = lee_c( fp ) ;
in.val   = lee_c( fp ) ;
in.pohor = lee_i( fp ) ;

return( in ) ;

}

/*-------------------------------------------------------------------------*/
/*        LEE_LB              Leer una lista de bucles y dar memoria                */
/*-------------------------------------------------------------------------*/

lbucl_s lee_lb ( FILE          *fp )        /* Fichero del que se va a leer */
{
                lbucl_s lis = LIST_0 ;      /* lista de bucles que se */
                                            /* va a rellenar */
                int       i ;               /* contador de bucle */
                int       j ;               /* valor intermedio */

j = lee_i( fp ) ;

lis = mem_lbucl( j, &lis ) ;
for( i=0 ; i<j ; i++ ) {
        lis.bucl[i] = lee_b( fp ) ;
        lis.n++ ;
        }

return( lis ) ;

}

/*-------------------------------------------------------------------------*/
/*        LEE_LBI             Leer una lista de bloques if y dar memoria          */
/*-------------------------------------------------------------------------*/

lblif_s lee_lbi ( FILE  *fp )           /* Fichero del que se va a leer */
{
                lblif_s lis = LIST_0 ;/* lista de bloques if que se */
                                            /* va a rellenar */
                int       i ;               /* contador de bucle */
                int       j ;               /* valor intermedio */

j = lee_i( fp ) ;
```

```
lis = mem_lblif( j, &lis ) ;
for( i=0 ; i<j ; i++ ) {
        lis.blif[i] = lee_bi( fp ) ;
        lis.n++ ;
        }

return( lis ) ;

}

/*-------------------------------------------------------------------*/
/*      LEE_LBL         Leer una lista de bloques y dar memoria      */
/*-------------------------------------------------------------------*/

lbloc_s lee_lbl ( FILE          *fp )           /* Fichero del que se va a leer */
{
                lbloc_s lis = LIST_0 ;/* lista de bloques que se */
                                      /* va a rellenar */
                int     i ;           /* contador de bucle */
                int     j ;           /* valor intermedio */

j = lee_i( fp ) ;

lis = mem_lbloc( j, &lis ) ;
for( i=0 ; i<j ; i++ ) {
        lis.bloc[i] = lee_bl( fp ) ;
        lis.n++ ;
        }

return( lis ) ;

}

/*-------------------------------------------------------------------*/
/*      LEE_LC          Leer una lista de cadenas y dar memoria      */
/*-------------------------------------------------------------------*/

lcad_s lee_lc ( FILE *fp )          /* Fichero del que se va a leer */

{
                lcad_s  lis = LIST_0 ;  /* lista de cadenas que se va */
                                        /* a rellenar */
                int     i ;             /* contador de bucle */
                int     j ;             /* valor intermedio */

j = lee_i( fp ) ;

lis = mem_lcad( j, &lis ) ;
for( i=0 ; i<j ; i++ ) {
        lis.cad[i] = lee_c( fp ) ;
        lis.n++ ;
        }

return( lis ) ;

}

/*-------------------------------------------------------------------*/
/*      LEE_LES         Leer una lista de entradas/salidas y dar memoria */
/*-------------------------------------------------------------------*/
```

```
lentsal_s lee_les ( FILE        *fp )               /* Fichero del que se lee */
{
                    lentsal_s       lis = LIST_0 ;      /* lista de entradas/salidas */
                                                        /* que se va a rellenar */
                    int             i ;                 /* contador de bucle */
                    int             j ;                 /* valor intermedio */

j = lee_i( fp ) ;

lis = mem_lentsal( j, &lis ) ;
for( i=0 ; i<j ; i++ ) {
        lis.ensa[i] = lee_es( fp ) ;
        lis.n++ ;
        }

return( lis ) ;

}

/*-------------------------------------------------------------------*/
/*      LEE_LF          Leer una lista de funciones y dar memoria    */
/*-------------------------------------------------------------------*/

lfunc_s lee_lf ( FILE           *fp )           /* Fichero del que se va a leer */
{
                    lfunc_s lis = LIST_0 ;      /* lista de funciones que se */
                                                /* va a rellenar */
                    int     i ;                 /* contador de bucle */
                    int     j ;                 /* valor intermedio */

j = lee_i( fp ) ;

lis = mem_lfunc( j, &lis ) ;
for( i=0 ; i<j ; i++ ) {
        lis.func[i] = lee1( fp ) ;
        lis.n++ ;
        }

return( lis ) ;

}

/*-------------------------------------------------------------------*/
/*      LEE_LIN         Leer una lista de inicializaciones y dar memoria */
/*-------------------------------------------------------------------*/

linic_s lee_lin ( FILE          *fp )           /* Fichero del que se va a leer */
{
                    linic_s lis = LIST_0 ;/* lista de inicializaciones que se */
                                                /* va a rellenar */
                    int     i ;                 /* contador de bucle */
                    int     j ;                 /* valor intermedio */

j = lee_i( fp ) ;

lis = mem_linic( j, &lis ) ;
for( i=0 ; i<j ; i++ ) {
        lis.inic[i] = lee_in( fp ) ;
        lis.n++ ;
        }
```

```
return( lis ) ;

}

/*-------------------------------------------------------------------*/
/*      LEE_LM          Leer una lista de macros y dar memoria       */
/*-------------------------------------------------------------------*/

lmacro_s lee_lm ( FILE          *fp )     /* Fichero del que se va a leer */
{
                lmacro_s lis = LIST_0 ;  /* lista de macros que se */
                                         /* va a rellenar */
                int      i ;             /* contador de bucle */
                int      j ;             /* valor intermedio */

j = lee_i( fp ) ;

lis = mem_lmacro( j, &lis ) ;
for( i=0 ; i<j ; i++ ) {
        lis.macro[i] = lee_m( fp ) ;
        lis.n++ ;
        }

return( lis ) ;

}

/*-------------------------------------------------------------------*/
/*      LEE_LOD         Leer una lista de origenes/destinos y dar memoria  */
/*-------------------------------------------------------------------*/

lorides_s lee_lod ( FILE      *fp )          /* Fichero del que se lee */
{
                lorides_s     lis = LIST_0 ;   /* lista de origenes/destinos*/
                                               /* que se va a rellenar */
                int           i ;              /* contador de bucle */
                int           j ;              /* valor intermedio */

j = lee_i( fp ) ;

lis = mem_lorides( j, &lis ) ;
for( i=0 ; i<j ; i++ ) {
        lis.od[i] = lee_od( fp ) ;
        lis.n++ ;
        }

return( lis ) ;

}

/*-------------------------------------------------------------------*/
/*      LEE_LR          Leer una lista de recursiones y dar memoria  */
/*-------------------------------------------------------------------*/

lrecu_s lee_lr ( FILE          *fp )     /* Fichero del que se va a leer */
{
                lrecu_s lis = LIST_0 ;   /* lista de recursiones que se */
                                         /* va a rellenar */
                int      i ;             /* contador de bucle */
                int      j ;             /* valor intermedio */
```

```
j = lee_i( fp ) ;

lis = mem_lrecu( j, &lis ) ;
for( i=0 ; i<j ; i++ ) {
        lis.recu[i] = lee_r( fp ) ;
        lis.n++ ;
        }

return( lis ) ;

}

/*-------------------------------------------------------------------*/
/*      LEE_M           Leer una estructura macro y dar memoria      */
/*-------------------------------------------------------------------*/

macro_s lee_m ( FILE        *fp )           /* Fichero del que se va a leer */
{
                macro_s  macro ;            /* estructura macro que se */
                                            /* va a rellenar */

macro.lfunc = lee_lf( fp ) ;
macro.lblif = lee_lbi( fp ) ;
macro.lbucl = lee_lb( fp ) ;
macro.lrecu = lee_lr( fp ) ;
macro.linic = lee_lin( fp ) ;

return( macro ) ;

}

/*-------------------------------------------------------------------*/
/*      LEE_OD          Leer una estructura origen/destino y dar memoria */
/*-------------------------------------------------------------------*/

orides_s lee_od ( FILE        *fp )   /* Fichero del que se va a leer */
{
                orides_s od ;               /* estructura origen/destino que se */
                                            /* va a rellenar */

od.nom   = NULL ;
od.ordes = NULL ;

od.nom   = lee_c( fp ) ;
od.ordes = lee_c( fp ) ;
od.tipo  = lee_uc( fp ) ;

return( od ) ;

}

/*-------------------------------------------------------------------*/
/*      LEE_R           Leer una estructura recursi¢n y dar memoria  */
/*-------------------------------------------------------------------*/

recu_s lee_r ( FILE *fp )               /* Fichero del que se va a leer */
{
                recu_s   recu ;             /* estructura recursi¢n que se */
                                            /* va a rellenar */
```

```
recu.fun   = NULL ;
recu.texto = NULL ;

recu.fun   = lee_c( fp ) ;
recu.texto = lee_c( fp ) ;

return( recu ) ;

}

/*-------------------------------------------------------------------*/
/*      LEE_UC              Leer un car cter                         */
/*-------------------------------------------------------------------*/

char lee_uc ( FILE  *fp )              /* Fichero del que se va a leer */
{

while( fgetc(fp) != ')' ) ;            /* el car cter debe ir precedido */
while( fgetc(fp) != '\0' ) ;           /* por:  ) \0  */

return( fgetc(fp) ) ;

}
```

```
/*-------------------------------------------------------------------------*/
/*      LEE.FUN          Funciones de lectura de fichero y carga en diagrama  */
/*-------------------------------------------------------------------------*/

/*      LEE              Leer un fichero *.ANA y dar memoria                  */

diag_s lee ( char   * ) ;          /* Identificador del fichero a leer */

/*-------------------------------------------------------------------------*/

/*      LEE1             Leer una estructura diagrama y dar memoria           */

diag_s lee1 ( FILE  * ) ;          /* Fichero del que se va a leer */

/*-------------------------------------------------------------------------*/

/*      LEE_B            Leer una estructura bucle y dar memoria              */

bucl_s lee_b ( FILE * ) ;          /* Fichero del que se va a leer */

/*-------------------------------------------------------------------------*/

/*      LEE_BI           Leer una estructura bloque if y dar memoria          */

blif_s lee_bi ( FILE  * ) ;        /* Fichero del que se va a leer */

/*-------------------------------------------------------------------------*/

/*      LEE_BL           Leer una estructura bloque y dar memoria             */

bloc_s lee_bl ( FILE* ) ;          /* Fichero del que se va a leer */

/*-------------------------------------------------------------------------*/

/*      LEE_C            Leer una cadena y reservarle memoria                 */

char *lee_c ( FILE  * ) ;          /* Fichero del que se va a leer */

/*-------------------------------------------------------------------------*/

/*      LEE_ES           Leer una estructura entrada/salida y dar memoria     */

entsal_s lee_es ( FILE      * ) ;  /* Fichero del que se va a leer */

/*-------------------------------------------------------------------------*/

/*      LEE_I            Leer un entero                                       */

int lee_i ( FILE    * ) ;          /* Fichero del que se va a leer */

/*-------------------------------------------------------------------------*/

/*      LEE_IN           Leer una estructura inicializaci¢n y dar memoria     */

inic_s lee_in ( FILE * ) ;         /* Fichero del que se va a leer */

/*-------------------------------------------------------------------------*/

/*      LEE_LB           Leer una lista de bucles y dar memoria               */

lbucl_s lee_lb ( FILE       * ) ;      /* Fichero del que se va a leer */
```

```
/*-------------------------------------------------------------------------*/
/*        LEE_LBI          Leer una lista de bloques if y dar memoria      */
lblif_s lee_lbi ( FILE  * ) ;          /* Fichero del que se va a leer */
/*-------------------------------------------------------------------------*/
/*        LEE_LBL          Leer una lista de bloques y dar memoria         */
lbloc_s lee_lbl ( FILE  * ) ;          /* Fichero del que se va a leer */
/*-------------------------------------------------------------------------*/
/*        LEE_LC           Leer una lista de cadenas y dar memoria         */
lcad_s lee_lc ( FILE * ) ;             /* Fichero del que se va a leer */
/*-------------------------------------------------------------------------*/
/*        LEE_LES          Leer una lista de entradas/salidas y dar memoria */
lentsal_s lee_les ( FILE  * ) ;        /* Fichero del que se lee */
/*-------------------------------------------------------------------------*/
/*        LEE_LF           Leer una lista de funciones y dar memoria       */
lfunc_s lee_lf ( FILE  * ) ;           /* Fichero del que se va a leer */
/*-------------------------------------------------------------------------*/
/*        LEE_LIN          Leer una lista de inicializaciones y dar memoria */
linic_s lee_lin ( FILE  * ) ;          /* Fichero del que se va a leer */
/*-------------------------------------------------------------------------*/
/*        LEE_LM           Leer una lista de macros y dar memoria          */
lmacro_s lee_lm ( FILE  * ) ;    /* Fichero del que se va a leer */
/*-------------------------------------------------------------------------*/
/*        LEE_LOD          Leer una lista de origenes/destinos y dar memoria */
lorides_s lee_lod ( FILE  * ) ;        /* Fichero del que se lee */
/*-------------------------------------------------------------------------*/
/*        LEE_LR           Leer una lista de recursiones y dar memoria     */
lrecu_s lee_lr ( FILE  * ) ;     /* Fichero del que se va a leer */
/*-------------------------------------------------------------------------*/
/*        LEE_M            Leer una estructura macro y dar memoria         */
macro_s lee_m ( FILE  * ) ;            /* Fichero del que se va a leer */
```

```
/*-------------------------------------------------------------------------*/
/*        LEE_OD          Leer una estructura origen/destino y dar memoria  */
orides_s lee_od ( FILE          * ) ;   /* Fichero del que se va a leer */
/*-------------------------------------------------------------------------*/
/*        LEE_R           Leer una estructura recursi¢n y dar memoria       */
recu_s lee_r ( FILE * ) ;               /* Fichero del que se va a leer */
/*-------------------------------------------------------------------------*/
/*        LEE_UC          Leer un car cter                                  */
char lee_uc ( FILE  * ) ;               /* Fichero del que se va a leer */
/*-------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------*/
/*        MEM_DIAG.C      REFERENTE A MEMORIA DE LA ESTRUCTURA DIAGRAMA        */
/*-------------------------------------------------------------------*/

#include <stdlib.h>
#include <stdio.h>

#include <diag.var>

#include <mem.fun>
#include <mem_diag.fun>

/*-------------------------------------------------------------------*/
/*   fre_blif                  Liberaci¢n de memoria de un bloque if          */
/*-------------------------------------------------------------------*/

blif_s fre_blif( blif_s                *lis )          /* Un bloque if */
{

lis->lbloc   = fre_lbloc( &(lis->lbloc) ) ;
lis->origenes = fre_lorides( &(lis->origenes) ) ;
                                                /*imp_mem() ;*/

return( *lis ) ;

}

/*-------------------------------------------------------------------*/
/*   fre_bloc                  Liberaci¢n de memoria de un bloque             */
/*-------------------------------------------------------------------*/

bloc_s fre_bloc( bloc_s                *lis )          /* Un bloque */
{

lis->fun   = fre_char( lis->fun ) ;
lis->con   = fre_char( lis->con ) ;
                                                /*imp_mem() ;*/

return( *lis ) ;

}

/*-------------------------------------------------------------------*/
/*   fre_bucl                  Liberaci¢n de memoria de un bucle              */
/*-------------------------------------------------------------------*/

bucl_s fre_bucl( bucl_s                *lis )          /* Un bucle */
{

lis->texaux = fre_char( lis->texaux ) ;
lis->texcon = fre_char( lis->texcon ) ;
lis->excon  = fre_char( lis->excon ) ;
lis->oc     = fre_lorides( &(lis->oc) ) ;
lis->ob     = fre_lorides( &(lis->ob) ) ;
lis->eb     = fre_lorides( &(lis->eb) ) ;
                                                /*imp_mem() ;*/

return( *lis ) ;

}
```

```
/*-------------------------------------------------------------------*/
/*   fre_diag                  Liberaci¢n de memoria de un diagrama o funci¢n          */
/*-------------------------------------------------------------------*/

diag_s fre_diag( diag_s                *lis )                    /* Un diagrama */
{

lis->id       = fre_char( lis->id ) ;
lis->nombre   = fre_lcad( &(lis->nombre) ) ;
lis->entradas = fre_lentsal( &(lis->entradas) ) ;
lis->salidas  = fre_lentsal( &(lis->salidas) ) ;
lis->interior = fre_lmacro( &(lis->interior) ) ;
                                                    /*imp_mem() ;*/

return( *lis ) ;

}

/*-------------------------------------------------------------------*/
/*   fre_entsal      Liberaci¢n de memoria de una entrada/salida      */
/*-------------------------------------------------------------------*/

entsal_s fre_entsal( entsal_s *lis )              /* Una entrada/salida */
{

lis->nombre = fre_char( lis->nombre ) ;
lis->orde = fre_lorides( &(lis->orde) ) ;
                                                    /*imp_mem() ;*/

return( *lis ) ;

}

/*-------------------------------------------------------------------*/
/*   fre_inic                  Liberaci¢n de memoria de una inicializaci¢n          */
/*-------------------------------------------------------------------*/

inic_s fre_inic( inic_s          *lis )           /* Una inicializaci¢n*/
{

lis->fun  = fre_char( lis->fun ) ;
lis->inp  = fre_char( lis->inp ) ;
lis->val  = fre_char( lis->val ) ;
                                                    /*imp_mem() ;*/

return( *lis ) ;

}

/*-------------------------------------------------------------------*/
/*   fre_lblif     Liberaci¢n de memoria de una lista de bloques if      */
/*-------------------------------------------------------------------*/

lblif_s fre_lblif( lblif_s         *lis )           /* Lista de bloques if */
{
                        int                i ;                    /* Contador de bucle */

if( lis->m > 0 ) {
          for( i=0 ; i < lis->n ; i++ )
                    lis->blif[i] = fre_blif( &(lis->blif[i]) ) ;
          free( lis->blif ) ;                /*imp_mem() ;*/
```

```
        }

lis->n  = 0 ;
lis->blif = NULL ;
lis->m  = 0 ;

return( *lis ) ;

}

/*-------------------------------------------------------------------------*/
/*   fre_lbloc     Liberaci¢n de memoria de una lista de bloques          */
/*-------------------------------------------------------------------------*/

lbloc_s fre_lbloc( lbloc_s     *lis )          /* Lista de bloques */
{
                int             i ;             /* Contador de bucle */

if( lis->m > 0 ) {
        for( i=0 ; i < lis->n ; i++ )
                lis->bloc[i] = fre_bloc( &(lis->bloc[i]) ) ;
        free( lis->bloc ) ;             /*imp_mem() ;*/
        }

lis->n  = 0 ;
lis->bloc = NULL ;
lis->m  = 0 ;

return( *lis ) ;

}

/*-------------------------------------------------------------------------*/
/*   fre_lbucl     Liberaci¢n de memoria de una lista de bucles           */
/*-------------------------------------------------------------------------*/

lbucl_s fre_lbucl( lbucl_s     *lis )          /* Lista de bucles */
{
                int             i ;             /* Contador de bucle */

if( lis->m > 0 ) {
        for( i=0 ; i < lis->n ; i++ )
                lis->bucl[i] = fre_bucl( &(lis->bucl[i]) ) ;
        free( lis->bucl ) ;             /*imp_mem() ;*/
        }

lis->n  = 0 ;
lis->bucl = NULL ;
lis->m  = 0 ;

return( *lis ) ;

}

/*-------------------------------------------------------------------------*/
/*   fre_lcad      Liberaci¢n de memoria de una lista de cadenas           */
/*-------------------------------------------------------------------------*/

lcad_s fre_lcad( lcad_s     *lis )          /* Lista de cadenas */
{
                int     i ;             /* Contador de bucle */
```

```
if( lis->m > 0 ) {
        for( i=0 ; i < lis->n ; i++ )
                lis->cad[i] = fre_char( lis->cad[i] ) ;
        free( lis->cad ) ;              /*imp_mem() ;*/
        }

lis->n = 0 ;
lis->cad = NULL ;
lis->m = 0 ;
return( *lis ) ;

}

/*-------------------------------------------------------------------------*/
/*   fre_lentsal  Liberaci¢n de memoria de una lista de entradas/salidas   */
/*-------------------------------------------------------------------------*/

lentsal_s fre_lentsal(          lentsal_s *lis )    /* Lista de ent/sal */
{
                                int                 i ;     /* Contador de bucle */

if( lis->m > 0 ) {
        for( i=0 ; i < lis->n ; i++ )
                lis->ensa[i] = fre_entsal( &(lis->ensa[i]) ) ;
        free( lis->ensa ) ;             /*imp_mem() ;*/
        }

lis->n  = 0 ;
lis->ensa = NULL ;
lis->m  = 0 ;

return( *lis ) ;

}

/*-------------------------------------------------------------------------*/
/*   fre_lfunc     Liberaci¢n de memoria de una lista de funciones     */
/*-------------------------------------------------------------------------*/

lfunc_s fre_lfunc( lfunc_s    *lis )            /* Lista de funciones */
{
                        int             i ;             /* Contador de bucle */

if( lis->m > 0 ) {
        for( i=0 ; i < lis->n ; i++ )
                lis->func[i] = fre_diag( &(lis->func[i]) ) ;
        free( lis->func ) ;             /*imp_mem() ;*/
        }

lis->n  = 0 ;
lis->func = NULL ;
lis->m  = 0 ;

return( *lis ) ;

}

/*-------------------------------------------------------------------------*/
/*   fre_linic     Liberaci¢n de memoria de una lista de inicializaciones  */
/*-------------------------------------------------------------------------*/
```

```
linic_s fre_linic( linic_s      *lis )              /* Lista de inicializaciones */
{
                    int             i ;                 /* Contador de bucle */

if( lis->m > 0 ) {
        for( i=0 ; i < lis->n ; i++ )
                lis->inic[i] = fre_inic( &(lis->inic[i]) ) ;
        free( lis->inic) ;              /*imp_mem() ;*/
        }

lis->n   = 0 ;
lis->inic = NULL ;
lis->m   = 0 ;

return( *lis ) ;

}

/*-------------------------------------------------------------------------*/
/*   fre_lmacro     Liberaci¢n de memoria de una lista de macros      */
/*-------------------------------------------------------------------------*/

lmacro_s fre_lmacro( lmacro_s       *lis )              /* Lista de macros */
{
                    int   i ;                   /* Contador de bucle */

if( lis->m > 0 ) {
        for( i=0 ; i < lis->n ; i++ )
                lis->macro[i] = fre_macro( &(lis->macro[i]) ) ;
        free( lis->macro) ;             /*imp_mem() ;*/
        }

lis->n   = 0 ;
lis->macro = NULL ;
lis->m   = 0 ;

return( *lis ) ;

}

/*-------------------------------------------------------------------------*/
/*   fre_lorides  Liberaci¢n de memoria de una lista de origenes/destinos   */
/*-------------------------------------------------------------------------*/

lorides_s fre_lorides(      lorides_s *lis )    /* Lista de or/des */
{
                    int             i ;         /* Contador de bucle */

if( lis->m > 0 ) {
        for( i=0 ; i < lis->n ; i++ )
                lis->od[i] = fre_orides( &(lis->od[i]) ) ;
        free( lis->od ) ;               /*imp_mem() ;*/
        }

lis->n   = 0 ;
lis->od = NULL ;
lis->m   = 0 ;

return( *lis ) ;
```

```
}

/*-------------------------------------------------------------------*/
/*   fre_lrecu     Liberaci¢n de memoria de una lista de recursiones   */
/*-------------------------------------------------------------------*/

lrecu_s fre_lrecu( lrecu_s     *lis )          /* Lista de recursiones */
{
                int             i ;             /* Contador de bucle */

if( lis->m > 0 ) {
        for( i=0 ; i < lis->n ; i++ )
                lis->recu[i] = fre_recu( &(lis->recu[i]) ) ;
        free( lis->recu ) ;             /*imp_mem() ;*/
        }

lis->n  = 0 ;
lis->recu = NULL ;
lis->m  = 0 ;

return( *lis ) ;

}

/*-------------------------------------------------------------------*/
/*   fre_macro             Liberaci¢n de memoria de una macro        */
/*-------------------------------------------------------------------*/

macro_s fre_macro( macro_s          *lis )              /* Una macro */
{

lis->lfunc  = fre_lfunc( &(lis->lfunc) ) ;
lis->lblif  = fre_lblif( &(lis->lblif) ) ;
lis->lbucl  = fre_lbucl( &(lis->lbucl) ) ;
lis->lrecu  = fre_lrecu( &(lis->lrecu) ) ;
lis->linic  = fre_linic( &(lis->linic) ) ;
                                        /*imp_mem() ;*/

return( *lis ) ;

}

/*-------------------------------------------------------------------*/
/*   fre_orides     Liberaci¢n de memoria de un origen/destino      */
/*-------------------------------------------------------------------*/

orides_s fre_orides( orides_s *lis )            /* Lista de or/des */
{

lis->nom = fre_char( lis->nom ) ;
lis->ordes = fre_char( lis->ordes ) ;
                                        /*imp_mem() ;*/
return( *lis ) ;

}

/*-------------------------------------------------------------------*/
/*   fre_recu              Liberaci¢n de memoria de una recursi¢n    */
/*-------------------------------------------------------------------*/

recu_s fre_recu( recu_s             *lis )          /* Una recursi¢n */
```

```
{

lis->fun   = fre_char( lis->fun ) ;
lis->texto = fre_char( lis->texto ) ;
                                                    /*imp_mem() ;*/

return( *lis ) ;

}

/*-------------------------------------------------------------------*/
/*   mem_lblif    Gesti¢n de memoria de una lista de bloques if      */
/*-------------------------------------------------------------------*/

lblif_s mem_lblif( int                  mem,            /* n£mero de elementos */
                   lblif_s  *lis )              /* lista de bloques if */
{

if( mem < = 0 ) {
        *lis = fre_lblif( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->blif = (blif_s *)calloc( mem, sizeof(blif_s) )) = = NULL ) {
                printf("\n Fallo al reservar %d blif_s", mem ) ;
                exit(1) ;
                }
        lis->n = 0 ;
        lis->m = mem ;          /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->blif = (blif_s *)realloc( lis->blif, mem * sizeof(blif_s) )) = = NULL ) {
                printf("\n Fallo al realocar %d blif_s", mem ) ;
                exit(1) ;
                }
        lis->m = mem ;          /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------*/
/*   mem_lbloc    Gesti¢n de memoria de una lista de bloques        */
/*-------------------------------------------------------------------*/

lbloc_s mem_lbloc( int                  mem,            /* n£mero de elementos */
                   lbloc_s *lis )               /* lista de bloques */
{
                   int                  i ;             /* contador de bucle */

if( mem < = 0 ) {
        *lis = fre_lbloc( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->bloc = (bloc_s *)calloc( mem, sizeof(bloc_s) )) = = NULL ) {
```

```
                printf("\n Fallo al reservar %d bloc_s", mem ) ;
                exit(1) ;
                }
        for( i=0 ; i<mem ; i++ ) {
                lis->bloc[i].fun = NULL ;
                lis->bloc[i].con = NULL ;
                }
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->bloc = (bloc_s *)realloc( lis->bloc, mem * sizeof(bloc_s) )) == NULL ) {
                printf("\n Fallo al realocar %d bloc_s", mem ) ;
                exit(1) ;
                }
        for( i=lis->m ; i<mem ; i++ ) {
                lis->bloc[i].fun = NULL ;
                lis->bloc[i].con = NULL ;
                }
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------*/
/*   mem_lbucl    Gesti¢n de memoria de una lista de bucles          */
/*-------------------------------------------------------------------*/

lbucl_s mem_lbucl( int              mem,            /* n£mero de elementos */
                   lbucl_s *lis )           /* lista de bloques if */
{
                   int              i ;             /* contador de bucle */

if( mem <= 0 ) {
        *lis = fre_lbucl( lis ) ; return( *lis ) ;
        }
if( mem == lis->m )
        return( *lis ) ;

if( lis->m == 0 ) {
        if( (lis->bucl = (bucl_s *)calloc( mem, sizeof(bucl_s) )) == NULL ) {
                printf("\n Fallo al reservar %d bucl_s", mem ) ;
                exit(1) ;
                }
        for( i=0 ; i<mem ; i++ ) {
                lis->bucl[i].texaux = NULL ;
                lis->bucl[i].texcon = NULL ;
                lis->bucl[i].excon = NULL ;
                }
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->bucl = (bucl_s *)realloc( lis->bucl, mem * sizeof(bucl_s) )) == NULL ) {
                printf("\n Fallo al realocar %d bucl_s", mem ) ;
                exit(1) ;
                }
        for( i=lis->m ; i<mem ; i++ ) {
```

```
                lis->bucl[i].texaux = NULL ;
                lis->bucl[i].texcon = NULL ;
                lis->bucl[i].excon = NULL ;
                }
        lis->m = mem ;          /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------*/
/*   mem_lcad      Gesti¢n de memoria de una lista de cadenas      */
/*-------------------------------------------------------------------*/

lcad_s mem_lcad( int        mem,            /* n£mero de elementos */
                 lcad_s *lis )              /* lista de cadenas */
{
                 int        i ;             /* contador de bucle */

if( mem < = 0 ) {
        *lis = fre_lcad( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->cad = (char **)calloc( mem, sizeof(char *) )) = = NULL ) {
                printf("\n Fallo al reservar %d char *", mem ) ;
                exit(1) ;
                }
        for( i=0 ; i<mem ; i++ )
                lis->cad[i] = NULL ;
        lis->n = 0 ;
        lis->m = mem ;          /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->cad = (char **)realloc( lis->cad, mem * sizeof(char *) )) = = NULL ) {
                printf("\n Fallo al realocar %d char *", mem ) ;
                exit(1) ;
                }
        for( i=lis->m ; i<mem ; i++ )
                lis->cad[i] = NULL ;
        lis->m = mem ;          /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------*/
/*   mem_lentsal   Gesti¢n de memoria de una lista de entradas/salidas  */
/*-------------------------------------------------------------------*/

lentsal_s mem_lentsal(  int             mem,    /* n£mero de elementos */
                        lentsal_s *lis )        /* lista de entradas/salidas */
{
                        int             i ;     /* contador de bucle */

if( mem < = 0 ) {
        *lis = fre_lentsal( lis ) ; return( *lis ) ;
        }
```

```
if( mem == lis->m )
        return( *lis ) ;

if( lis->m == 0 ) {
        if( (lis->ensa = (entsal_s *)calloc( mem, sizeof(entsal_s) )) == NULL ) {
                printf("\n Fallo al reservar %d entsal_s", mem ) ;
                exit(1) ;
                }
        for( i=0 ; i<mem ; i++ )
                lis->ensa[i].nombre = NULL ;
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->ensa = (entsal_s *)realloc( lis->ensa, mem * sizeof(entsal_s) )) == NULL ) {
                printf("\n Fallo al realocar %d entsal_s", mem ) ;
                exit(1) ;
                }
        for( i=lis->m ; i<mem ; i++ )
                lis->ensa[i].nombre = NULL ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------*/
/*   mem_lfunc    Gesti¢n de memoria de una lista de funciones      */
/*-------------------------------------------------------------------*/

lfunc_s mem_lfunc( int              mem,            /* n£mero de elementos */
                   lfunc_s *lis )           /* lista de funciones */
{
                   int              i ;        /* Contador de bucle */

if( mem <= 0 ) {
        *lis = fre_lfunc( lis ) ; return( *lis ) ;
        }
if( mem == lis->m )
        return( *lis ) ;

if( lis->m == 0 ) {
        if( (lis->func = (diag_s *)calloc( mem, sizeof(diag_s) )) == NULL ) {
                printf("\n Fallo al reservar %d diag_s", mem ) ;
                exit(1) ;
                }
        for( i=0 ; i<mem ; i++ )
                lis->func[i].id = NULL ;
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->func = (diag_s *)realloc( lis->func, mem * sizeof(diag_s) )) == NULL ) {
                printf("\n Fallo al realocar %d diag_s", mem ) ;
                exit(1) ;
                }
        for( i=lis->m ; i<mem ; i++ )
                lis->func[i].id = NULL ;
        lis->m = mem ;              /*imp_mem() ;*/
```

```
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------*/
/*   mem_linic    Gesti¢n de memoria de una lista de inicializaciones   */
/*-------------------------------------------------------------------*/

linic_s mem_linic( int                  mem,            /* n£mero de elementos */
                   linic_s  *lis )              /* lista de inicializaciones */
{
                   int              i ;                 /* contador de bucle */

if( mem < = 0 ) {
        *lis = fre_linic( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->inic = (inic_s *)calloc( mem, sizeof(inic_s) )) = = NULL ) {
                printf("\n Fallo al reservar %d inic_s", mem ) ;
                exit(1) ;
                }
        for( i=0 ; i<mem ; i++ ) {
                lis->inic[i].fun = NULL ;
                lis->inic[i].inp = NULL ;
                lis->inic[i].val = NULL ;
                }
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->inic = (inic_s *)realloc( lis->inic, mem * sizeof(inic_s) )) = = NULL ) {
                printf("\n Fallo al realocar %d inic_s", mem ) ;
                exit(1) ;
                }
        for( i=lis->m ; i<mem ; i++ ) {
                lis->inic[i].fun = NULL ;
                lis->inic[i].inp = NULL ;
                lis->inic[i].val = NULL ;
                }
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------*/
/*   mem_lmacro    Gesti¢n de memoria de una lista de macros      */
/*-------------------------------------------------------------------*/

lmacro_s mem_lmacro(    int             mem,    /* n£mero de elementos */
                        lmacro_s *lis )     /* lista de macros */
{

if( mem < = 0 ) {
        *lis = fre_lmacro( lis ) ; return( *lis ) ;
        }
```

```
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->macro = (macro_s *)calloc( mem, sizeof(macro_s) )) = = NULL ) {
                printf("\n Fallo al reservar %d macro_s", mem ) ;
                exit(1) ;
                }
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->macro = (macro_s *)realloc( lis->macro, mem * sizeof(macro_s) )) = = NULL ) {
                printf("\n Fallo al realocar %d macro_s", mem ) ;
                exit(1) ;
                }
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------------*/
/*   mem_lorides  Gesti¢n de memoria de una lista de origenes/destinos     */
/*-------------------------------------------------------------------------*/

lorides_s mem_lorides(      int             mem,    /* n£mero de elementos */
                            lorides_s *lis )    /* lista de origenes/destinos*/
{
                            int             i ;     /* contador de bucle */

if( mem < = 0 ) {
        *lis = fre_lorides( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->od = (orides_s *)calloc( mem, sizeof(orides_s) )) = = NULL ) {
                printf("\n Fallo al reservar %d orides_s", mem ) ;
                exit(1) ;
                }
        for( i=0 ; i<mem ; i++ ) {
                lis->od[i].nom = NULL ;
                lis->od[i].ordes = NULL ;
                }
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->od = (orides_s *)realloc( lis->od, mem * sizeof(orides_s) )) = = NULL ) {
                printf("\n Fallo al realocar %d orides_s", mem ) ;
                exit(1) ;
                }
        for( i=lis->m ; i<mem ; i++ ) {
                lis->od[i].nom = NULL ;
                lis->od[i].ordes = NULL ;
                }
        lis->m = mem ;              /*imp_mem() ;*/
```

```
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------------*/
/*   mem_lrecu    Gesti¢n de memoria de una lista de recursiones           */
/*-------------------------------------------------------------------------*/

lrecu_s mem_lrecu( int              mem,            /* n£mero de elementos */
                   lrecu_s *lis )           /* lista de recursiones */
{
                   int              i ;             /* contador de bucle */

if( mem < = 0 ) {
        *lis = fre_lrecu( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->recu = (recu_s *)calloc( mem, sizeof(recu_s) )) = = NULL ) {
                printf("\n Fallo al reservar %d recu_s", mem ) ;
                exit(1) ;
                }
        for( i=0 ; i<mem ; i+ + ) {
                lis->recu[i].fun = NULL ;
                lis->recu[i].texto = NULL ;
                }
        lis->n = 0 ;
        lis->m = mem ;          /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->recu = (recu_s *)realloc( lis->recu, mem * sizeof(recu_s) )) = = NULL ) {
                printf("\n Fallo al realocar %d recu_s", mem ) ;
                exit(1) ;
                }
        for( i=lis->m ; i<mem ; i+ + ) {
                lis->recu[i].fun = NULL ;
                lis->recu[i].texto = NULL ;
                }
        lis->m = mem ;          /*imp_mem() ;*/
        return( *lis ) ;
        }

}
```

```
/*-------------------------------------------------------------------*/
/*        MEM_DIAG        Funciones de memoria de la estructura diag_s      */
/*-------------------------------------------------------------------*/

/*        FRE_BLIF Liberar memoria de un bloque if */

blif_s fre_blif ( blif_s * ) ;              /* bloque if */

/*-------------------------------------------------------------------*/

/*        FRE_BLOC        Liberaci¢n de memoria de un bloque              */

bloc_s fre_bloc( bloc_s      * ) ;          /* Un bloque */

/*-------------------------------------------------------------------*/

/*        FRE_BUCL        Liberar memoria de un bucle */

bucl_s fre_bucl ( bucl_s * ) ;          /* bucle */

/*-------------------------------------------------------------------*/

/*        FRE_DIAG        Liberar memoria de un diagrama */

diag_s fre_diag ( diag_s * ) ;          /* diagrama */

/*-------------------------------------------------------------------*/

/*        FRE_ENTSAL      Liberar memoria de una entrada/salida */

entsal_s fre_entsal ( entsal_s * ) ;            /* entrada/salida */

/*-------------------------------------------------------------------*/

/*        FRE_INIC Liberar memoria de una inicializaci¢n */

inic_s fre_inic ( inic_s * ) ;          /* inicializaci¢n */

/*-------------------------------------------------------------------*/

/*        FRE_LBLIF       Liberar memoria de una lista de bloque if */

lblif_s fre_lblif ( lblif_s * ) ;           /* lista de bloque if */

/*-------------------------------------------------------------------*/

/*   FRE_LBLOC    Liberaci¢n de memoria de una lista de bloques          */

lbloc_s fre_lbloc(  lbloc_s     * ) ;          /* Lista de bloques */

/*-------------------------------------------------------------------*/

/*        FRE_LBUCL       Liberar memoria de una lista de bucle */

lbucl_s fre_lbucl ( lbucl_s * ) ;               /* lista de bucle */

/*-------------------------------------------------------------------*/

/*        FRE_LCAD        Liberar memoria en una lista de cadenas */
```

```
lcad_s fre_lcad ( lcad_s * ) ;          /* Lista de cadenas */
/*-------------------------------------------------------------------*/
/*      FRE_LENTSAL     Liberar memoria en una lista de entrada/salida */
lentsal_s fre_lentsal ( lentsal_s * ) ;          /* lista de entrada/salida */
/*-------------------------------------------------------------------*/
/*      FRE_LFUNC       Liberar memoria de una lista de funciones */
lfunc_s fre_lfunc ( lfunc_s * ) ;          /* lista de funciones */
/*-------------------------------------------------------------------*/
/*      FRE_LINIC       Liberar memoria de una lista de inicializaci¢n */
linic_s fre_linic ( linic_s * ) ;          /* lista de inicializaci¢n */
/*-------------------------------------------------------------------*/
/*      FRE_LMACRO      Liberar memoria de una lista de macro */
lmacro_s fre_lmacro ( lmacro_s * ) ;          /* lista de macro */
/*-------------------------------------------------------------------*/
/*      FRE_LORIDES     Liberar memoria en una lista de origen/destino */
lorides_s fre_lorides ( lorides_s * ) ;          /* lista de origen/destino */
/*-------------------------------------------------------------------*/
/*      FRE_LRECUR      Liberar memoria de una lista de recursi¢n */
lrecu_s fre_lrecu ( lrecu_s * ) ;          /* lista de recursi¢n */
/*-------------------------------------------------------------------*/
/*      FRE_MACRO       Liberar memoria de una macro */
macro_s fre_macro ( macro_s * ) ;          /* macro */
/*-------------------------------------------------------------------*/
/*      FRE_ORIDES      Liberar memoria de un origen/destino */
orides_s fre_orides ( orides_s * ) ;          /* origen/destino */
/*-------------------------------------------------------------------*/
/*      FRE_RECUR       Liberar memoria de una recursi¢n */
recu_s fre_recu ( recu_s * ) ;          /* recursi¢n */
/*-------------------------------------------------------------------*/
/*      MEM_LBLIF       Reservar memoria en una lista de bloques if */
lblif_s mem_lblif ( int,          /* memoria que se quiere reservar */
```

```
                        lblif_s * ) ;       /* lista de bloques if */

/*-------------------------------------------------------------------*/

/*   MEM_LBLOC    Gesti¢n de memoria de una lista de bloques          */

lbloc_s mem_lbloc( int        ,                   /* n£mero de elementos */
                        lbloc_s * ) ;       /* lista de bloques */

/*-------------------------------------------------------------------*/

/*        MEM_LBUCL       Reservar memoria en una lista de bucles */

lbucl_s mem_lbucl ( int,                    /* memoria que se quiere reservar */
                        lbucl_s * ) ;       /* lista de bucles */

/*-------------------------------------------------------------------*/

/*     MEM_LCAD Reservar memoria en una lista de cadenas */

lcad_s mem_lcad ( int,                      /* memoria que se quiere reservar */
                        lcad_s * ) ;        /* lista de cadenas en cuesti¢n */

/*-------------------------------------------------------------------*/

/*        MEM_LENTSAL     Reservar memoria en una lista de entradas/salidas */

lentsal_s mem_lentsal ( int,                /* memoria que se quiere reservar */
                        lentsal_s * ) ; /* lista de entradas/salidas */

/*-------------------------------------------------------------------*/

/*        MEM_LFUNC       Reservar memoria en una lista de funciones */

lfunc_s mem_lfunc ( int,                    /* memoria que se quiere reservar */
                        lfunc_s * ) ;       /* lista de funciones */

/*-------------------------------------------------------------------*/

/*        MEM_LINIC       Reservar memoria en una lista de inicializaciones */

linic_s mem_linic ( int,                    /* memoria que se quiere reservar */
                        linic_s * ) ;       /* lista de inicializaciones */

/*-------------------------------------------------------------------*/

/*        MEM_LMACRO      Reservar memoria en una lista de macros */

lmacro_s mem_lmacro ( int,                  /* memoria que se quiere reservar */
                        lmacro_s * ) ; /* lista de macros */

/*-------------------------------------------------------------------*/

/*        MEM_LORIDES     Reservar memoria en una lista de origenes/destinos */

lorides_s mem_lorides ( int,                /* memoria que se quiere reservar */
                        lorides_s * ) ; /* lista de origenes/destinos */

/*-------------------------------------------------------------------*/

/*        MEM_LRECUR      Reservar memoria en una lista de recursiones */
```

```
lrecu_s mem_lrecu ( int,                /* memoria que se quiere reservar */
                    lrecu_s * ) ;       /* lista de recursiones */

/*-------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------------*/
/*        MEM_DIBU.C      REFERENTE A MEMORIA DE LA ESTRUCTURA DIBUJO       */
/*-------------------------------------------------------------------------*/

#include <stdlib.h>
#include <stdio.h>

#include <dibu.var>

#include <mem.fun>
#include <mem_dibu.fun>

/*-------------------------------------------------------------------------*/
/*   fre_conjunto   Liberaci¢n de memoria de una estructura conjunto    */
/*-------------------------------------------------------------------------*/

conjunto_s fre_conjunto( conjunto_s   *lis )      /* Una estructura conjunto */
{

lis->dimfun = fre_dimfun( &(lis->dimfun) ) ;
lis->dimhue = fre_dimhue( &(lis->dimhue) ) ;
lis->relele = fre_relele( &(lis->relele) ) ;
lis->lgrupo = fre_lgrupo( &(lis->lgrupo) ) ;
                                          /*imp_mem() ;*/

return( *lis ) ;

}

/*-------------------------------------------------------------------------*/
/*   fre_dibu                 Liberaci¢n de memoria de un dibujo                */
/*-------------------------------------------------------------------------*/

dibujo_s fre_dibu( dibujo_s   *lis )              /* Un bloque if */
{

lis->inter    = fre_inter( &(lis->inter) ) ;
lis->conjunto = fre_conjunto( &(lis->conjunto) ) ;
                                          /*imp_mem() ;*/

return( *lis ) ;

}

/*-------------------------------------------------------------------------*/
/*   fre_dimfun  Liberaci¢n de memoria de dimensi¢n de funciones             */
/*-------------------------------------------------------------------------*/

dimfun_s fre_dimfun( dimfun_s         *lis )          /* Una estructura de dimensi¢n de funciones */
{

lis->lfunhor = fre_lfunhor( &(lis->lfunhor) ) ;
lis->lfunver = fre_lfunver( &(lis->lfunver) ) ;
                                          /*imp_mem() ;*/

return( *lis ) ;

}

/*-------------------------------------------------------------------------*/
/*   fre_dimhue  Liberaci¢n de memoria de dimensi¢n de huecos                */
```

```
/*-------------------------------------------------------------------*/

dimhue_s fre_dimhue( dimhue_s        *lis )              /* Una estructura de dimensi¢n de huecos */
{

lis->lhueh = fre_lhueh( &(lis->lhueh) ) ;
lis->lhuev = fre_lhuev( &(lis->lhuev) ) ;
                                                /*imp_mem() ;*/

return( *lis ) ;

}

/*-------------------------------------------------------------------*/
/*   fre_fledat      Liberaci¢n de memoria de una flecha dato        */
/*-------------------------------------------------------------------*/

fledat_s fre_fledat( fledat_s  *lis )              /* Una flecha dato */
{

lis->ori = fre_liop( &(lis->ori) ) ;
lis->des = fre_liop( &(lis->des) ) ;
                                                /*imp_mem() ;*/

return( *lis ) ;

}

/*-------------------------------------------------------------------*/
/*   fre_flecha      Liberaci¢n de memoria de una flecha             */
/*-------------------------------------------------------------------*/

flecha_s fre_flecha( flecha_s *lis )               /* Una flecha */
{

lis->lch = fre_lcoohor( &(lis->lch) ) ;
lis->lcv = fre_lcoover( &(lis->lcv) ) ;
                                                /*imp_mem() ;*/

return( *lis ) ;

}

/*-------------------------------------------------------------------*/
/*   fre_grupo       Liberaci¢n de memoria de un grupo               */
/*-------------------------------------------------------------------*/

grupo_s fre_grupo( grupo_s  *lis )                 /* Un grupo */
{

lis->conjunto = fre_conjunto( &(lis->conjunto) ) ;
                                                /*imp_mem() ;*/

return( *lis ) ;

}

/*-------------------------------------------------------------------*/
/*   fre_inpout      Liberaci¢n de memoria de un inpout              */
/*-------------------------------------------------------------------*/
```

```
inpout_s fre_inpout( inpout_s          *lis )          /* Un inpout */
{

lis->nom = fre_char( lis->nom ) ;
                                                /*imp_mem() ;*/

return( *lis ) ;

}

/*-------------------------------------------------------------------*/
/*   fre_inter       Liberaci¢n de memoria de una estructura inter    */
/*-------------------------------------------------------------------*/

inter_s fre_inter( inter_s      *lis )          /* Una estructura inter */
{

lis->entradas   = fre_liof( &(lis->entradas) ) ;
lis->salidas    = fre_liof( &(lis->salidas) ) ;
lis->lfledat = fre_lfledat( &(lis->lfledat) ) ;
lis->linterf = fre_linterf( &(lis->linterf) ) ;
                                                /*imp_mem() ;*/

return( *lis ) ;

}

/*-------------------------------------------------------------------*/
/*   fre_interf  Liberaci¢n de memoria de una estructura inter con funci¢n */
/*-------------------------------------------------------------------*/

interf_s fre_interf( interf_s   *lis )          /* Una estructura interf */
{

lis->fun   = fre_char( lis->fun ) ;
lis->inter = fre_inter( &(lis->inter) ) ;
                                                /*imp_mem() ;*/

return( *lis ) ;

}

/*-------------------------------------------------------------------*/
/*   fre_iof         Liberaci¢n de memoria de un inpout con funci¢n   */
/*-------------------------------------------------------------------*/

iof_s fre_iof( iof_s  *lis )            /* Un inpout con funci¢n */
{

lis->fun    = fre_char( lis->fun ) ;
lis->inpout = fre_inpout( &(lis->inpout) ) ;
                                                /*imp_mem() ;*/

return( *lis ) ;

}
/*-------------------------------------------------------------------*/
/*   fre_larco       Liberaci¢n de memoria de una lista de arcos      */
/*-------------------------------------------------------------------*/

larco_s fre_larco( larco_s      *lis )          /* Lista de arcos */
```

```
{

if( lis->m > 0 ) {
        free( lis->arco) ;              /*imp_mem() ;*/
        }

lis->n   = 0 ;
lis->arco = NULL ;
lis->m   = 0 ;

return( *lis ) ;

}
/*-------------------------------------------------------------------------*/
/*    fre_lcoohor Liberaci¢n de memoria de una lista de coordenadas horizon. */
/*-------------------------------------------------------------------------*/

lcoohor_s fre_lcoohor( lcoohor_s *lis )          /* Lista de coordenadas horizontales */
{

if( lis->m > 0 ) {
        free( lis->coohor) ;                    /*imp_mem() ;*/
        }

lis->n   = 0 ;
lis->coohor = NULL ;
lis->m   = 0 ;

return( *lis ) ;

}
/*-------------------------------------------------------------------------*/
/*    fre_lcoover Liberaci¢n de memoria de una lista de coordenadas vertica. */
/*-------------------------------------------------------------------------*/

lcoover_s fre_lcoover( lcoover_s *lis )          /* Lista de coordenadas verticales */
{

if( lis->m > 0 ) {
        free( lis->coover) ;                    /*imp_mem() ;*/
        }

lis->n   = 0 ;
lis->coover = NULL ;
lis->m   = 0 ;

return( *lis ) ;

}
/*-------------------------------------------------------------------------*/
/*    fre_lflecha  Liberaci¢n de memoria de una lista de flechas           */
/*-------------------------------------------------------------------------*/

lflecha_s fre_lflecha( lflecha_s        *lis )   /* Lista de flechas  */
{
                int             i ;     /* Contador de bucle */

if( lis->m > 0 ) {
        for( i=0 ; i < lis->n ; i++ )
                lis->flecha[i] = fre_flecha( &(lis->flecha[i]) ) ;
        free( lis->flecha) ;            /*imp_mem() ;*/
```

```
        }

lis->n  = 0 ;
lis->flecha = NULL ;
lis->m  = 0 ;

return( *lis ) ;

}
/*-------------------------------------------------------------------*/
/*   fre_lfledat  Liberaci¢n de memoria de una lista de flechas dato  */
/*-------------------------------------------------------------------*/

lfledat_s fre_lfledat( lfledat_s        *lis )    /* Lista de flechas dato */
{
                        int             i ;       /* Contador de bucle */

if( lis->m > 0 ) {
        for( i=0 ; i < lis->n ; i++ )
                lis->fledat[i] = fre_fledat( &(lis->fledat[i]) ) ;
        free( lis->fledat) ;            /*imp_mem() ;*/
        }

lis->n  = 0 ;
lis->fledat = NULL ;
lis->m  = 0 ;

return( *lis ) ;

}
/*-------------------------------------------------------------------*/
/*  fre_lfunhor Liberaci¢n de memoria de una lista de funciones horizontales */
/*-------------------------------------------------------------------*/

lfunhor_s fre_lfunhor( lfunhor_s *lis )         /* Lista de funciones horizontales (dimensi¢n) */
{

if( lis->m > 0 ) {
        free( lis->funhor) ;            /*imp_mem() ;*/
        }

lis->n  = 0 ;
lis->funhor = NULL ;
lis->m  = 0 ;

return( *lis ) ;

}
/*-------------------------------------------------------------------*/
/*  fre_lfunver Liberaci¢n de memoria de una lista de funciones verticales  */
/*-------------------------------------------------------------------*/

lfunver_s fre_lfunver( lfunver_s *lis )         /* Lista de funciones verticales (dimensi¢n) */
{

if( lis->m > 0 ) {
        free( lis->funver) ;            /*imp_mem() ;*/
        }

lis->n  = 0 ;
lis->funver = NULL ;
```

```
lis->m  = 0 ;

return( *lis ) ;

}
/*-------------------------------------------------------------------*/
/*    fre_lgrupo  Liberaci¢n de memoria de una lista de grupos       */
/*-------------------------------------------------------------------*/

lgrupo_s fre_lgrupo( lgrupo_s          *lis )          /* Lista de grupos */
{
                    int    i ;                 /* Contador de bucle */

if( lis->m > 0 ) {
          for( i=0 ; i < lis->n ; i++ )
                    lis->grupo[i] = fre_grupo( &(lis->grupo[i]) ) ;
          free( lis->grupo) ;          /*imp_mem() ;*/
          }

lis->n  = 0 ;
lis->grupo = NULL ;
lis->m  = 0 ;

return( *lis ) ;

}
/*-------------------------------------------------------------------*/
/*    fre_lhueh    Liberaci¢n de memoria de una lista de huecos horizontales */
/*-------------------------------------------------------------------*/

lhueh_s fre_lhueh( lhueh_s *lis )          /* Lista de huecos horizontales (dimensi¢n) */
{

if( lis->m > 0 ) {
          free( lis->hueh) ;          /*imp_mem() ;*/
          }

lis->n  = 0 ;
lis->hueh = NULL ;
lis->m  = 0 ;

return( *lis ) ;

}
/*-------------------------------------------------------------------*/
/*    fre_lhuev    Liberaci¢n de memoria de una lista de huecos verticales   */
/*-------------------------------------------------------------------*/

lhuev_s fre_lhuev( lhuev_s *lis )          /* Lista de huecos verticales (dimensi¢n) */
{

if( lis->m > 0 ) {
          free( lis->huev) ;          /*imp_mem() ;*/
          }

lis->n  = 0 ;
lis->huev = NULL ;
lis->m  = 0 ;

return( *lis ) ;
```

```
}

/*-------------------------------------------------------------------------*/
/*   fre_linea        Liberaci¢n de memoria de una estructura linea      */
/*-------------------------------------------------------------------------*/

linea_s fre_linea( linea_s      *lis )              /* Una estructura linea */
{

lis->x0    = fre_lcoohor( &(lis->x0) ) ;
lis->x1    = fre_lcoover( &(lis->x1) ) ;
lis->y0    = fre_lcoohor( &(lis->y0) ) ;
lis->y1    = fre_lcoover( &(lis->y1) ) ;
                                                /*imp_mem() ;*/

return( *lis ) ;

}
/*-------------------------------------------------------------------------*/
/*   fre_linterf  Liberaci¢n de memoria de una lista de inter con funci¢n   */
/*-------------------------------------------------------------------------*/

linterf_s fre_linterf(  linterf_s *lis )      /* Lista de inter con funci¢n */
{
                        int                 i ;         /* Contador de bucle */

if( lis->m > 0 ) {
        for( i=0 ; i < lis->n ; i++ )
                lis->interf[i] = fre_interf( &(lis->interf[i]) ) ;
        free( lis->interf) ;            /*imp_mem() ;*/
        }

lis->n  = 0 ;
lis->interf = NULL ;
lis->m  = 0 ;

return( *lis ) ;

}
/*-------------------------------------------------------------------------*/
/*   fre_liof  Liberaci¢n de memoria de una lista de inpouts con funci¢n   */
/*-------------------------------------------------------------------------*/

liof_s fre_liof(  liof_s          *lis )              /* Lista de inpouts con funci¢n */
{
                int             i ;                 /* Contador de bucle */

if( lis->m > 0 ) {
        for( i=0 ; i < lis->n ; i++ )
                lis->iof[i] = fre_iof( &(lis->iof[i]) ) ;
        free( lis->iof) ;               /*imp_mem() ;*/
        }

lis->n  = 0 ;
lis->iof = NULL ;
lis->m  = 0 ;

return( *lis ) ;

}
/*-------------------------------------------------------------------------*/
```

```
/*    fre_liop  Liberaci¢n de memoria de una lista de inpouts puntero          */
/*-------------------------------------------------------------------------*/

liop_s fre_liop( liop_s          *lis )          /* Lista de inpouts puntero */
{

if( lis->m > 0 ) {
          free( lis->inpout) ;          /*imp_mem() ;*/
          }

lis->n  = 0 ;
lis->inpout = NULL ;
lis->m  = 0 ;

return( *lis ) ;

}
/*-------------------------------------------------------------------------*/
/*    fre_llindo  Liberaci¢n de memoria de una lista de l¡neas dobles          */
/*-------------------------------------------------------------------------*/

llindo_s fre_llindo( llindo_s    *lis )          /* Lista de l¡neas dobles */
{

if( lis->m > 0 ) {
          free( lis->lindo) ;          /*imp_mem() ;*/
          }

lis->n  = 0 ;
lis->lindo = NULL ;
lis->m  = 0 ;

return( *lis ) ;

}
/*-------------------------------------------------------------------------*/
/*    fre_llinea  Liberaci¢n de memoria de una lista de l¡neas                */
/*-------------------------------------------------------------------------*/

llinea_s fre_llinea( llinea_s    *lis )          /* Lista de l¡neas */
{
                    int    i ;                /* Contador de bucle */

if( lis->m > 0 ) {
          for( i=0 ; i < lis->n ; i++ )
                    lis->linea[i] = fre_linea( &(lis->linea[i]) ) ;
          free( lis->linea) ;          /*imp_mem() ;*/
          }

lis->n  = 0 ;
lis->linea = NULL ;
lis->m  = 0 ;

return( *lis ) ;

}
/*-------------------------------------------------------------------------*/
/*    fre_lmarca  Liberaci¢n de memoria de una lista de marca                 */
/*-------------------------------------------------------------------------*/

lmarca_s fre_lmarca( lmarca_s          *lis )          /* Lista de marcas */
```

```
{

if( lis->m > 0 ) {
        free( lis->marca) ;          /*imp_mem() ;*/
        }

lis->n  = 0 ;
lis->marca = NULL ;
lis->m  = 0 ;

return( *lis ) ;

}
/*-------------------------------------------------------------------*/
/*   fre_lrect  Liberaci¢n de memoria de una lista de rectangulos    */
/*-------------------------------------------------------------------*/

lrect_s fre_lrect( lrect_s     *lis )          /* Lista de rectangulos */
{

if( lis->m > 0 ) {
        free( lis->rect) ;           /*imp_mem() ;*/
        }

lis->n  = 0 ;
lis->rect = NULL ;
lis->m  = 0 ;

return( *lis ) ;

}
/*-------------------------------------------------------------------*/
/*   fre_ltexele    Liberaci¢n de memoria de una lista de textos     */
/*-------------------------------------------------------------------*/

ltexele_s fre_ltexele( ltexele_s *lis )          /* Lista de textos */
{
                int   i ;                        /* Contador de bucle */

if( lis->m > 0 ) {
        for( i=0 ; i < lis->n ; i++ )
                lis->texele[i] = fre_texele( &(lis->texele[i]) ) ;
        free( lis->texele) ;         /*imp_mem() ;*/
        }

lis->n  = 0 ;
lis->texele = NULL ;
lis->m  = 0 ;

return( *lis ) ;

}

/*-------------------------------------------------------------------*/
/*   fre_relele      Liberaci¢n de memoria de una estructura relele  */
/*-------------------------------------------------------------------*/

relele_s fre_relele( relele_s   *lis )           /* Una estructura relele */
{

lis->text    = fre_texdib( &(lis->text) ) ;
```

```
lis->lrect    = fre_lrect( &(lis->lrect) ) ;
lis->larcf = fre_larco( &(lis->larcf) ) ;
lis->larcg = fre_larco( &(lis->larcg) ) ;
lis->llinf = fre_llinea( &(lis->llinf) ) ;
lis->lling = fre_llinea( &(lis->lling) ) ;
lis->llindo = fre_llindo( &(lis->llindo) ) ;
lis->lflecha = fre_lflecha( &(lis->lflecha) ) ;
lis->lmarca = fre_lmarca( &(lis->lmarca) ) ;
                                                /*imp_mem() ;*/

return( *lis ) ;

}

/*-------------------------------------------------------------------------*/
/*   fre_texdib      Liberaci¢n de memoria de una estructura texdib        */
/*-------------------------------------------------------------------------*/

texdib_s fre_texdib( texdib_s *lis )            /* Una estructura texdib */
{

lis->ltdat = fre_ltexele( &(lis->ltdat) ) ;
lis->ltfun = fre_ltexele( &(lis->ltfun) ) ;
lis->ltidc = fre_ltexele( &(lis->ltidc) ) ;
lis->ltidp = fre_ltexele( &(lis->ltidp) ) ;
lis->tt    = fre_texele( &(lis->tt) ) ;
                                                /*imp_mem() ;*/

return( *lis ) ;

}

/*-------------------------------------------------------------------------*/
/*   fre_texele      Liberaci¢n de memoria de un texto                     */
/*-------------------------------------------------------------------------*/

texele_s fre_texele( texele_s *lis )            /* Una estructura texele */
{

lis->s    = fre_char( lis->s ) ;
                                                /*imp_mem() ;*/

return( *lis ) ;

}

/*-------------------------------------------------------------------------*/
/*   mem_larco     Gesti¢n de memoria de una lista de arcos                */
/*-------------------------------------------------------------------------*/

larco_s mem_larco( int                  mem,            /* n£mero de elementos */
                   larco_s *lis )               /* lista de arco */
{

if( mem < = 0 ) {
        *lis = fre_larco( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
```

```
        if( (lis->arco = (arco_s *)calloc( mem, sizeof(arco_s) )) == NULL ) {
                printf("\n Fallo al reservar %d arco_s *", mem ) ;
                exit(1) ;
                }
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->arco = (arco_s *)realloc( lis->arco, mem * sizeof(arco_s) )) == NULL ) {
                printf("\n Fallo al realocar %d arco_s *", mem ) ;
                exit(1) ;
                }
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------*/
/*  mem_lcoohor  Gesti¢n de memoria de una lista de coordenadas horizontales */
/*-------------------------------------------------------------------*/

lcoohor_s mem_lcoohor( int  mem,             /* n£mero de elementos */
                       lcoohor_s *lis )              /* lista de coordenadas horizontales */
{

if( mem < = 0 ) {
        *lis = fre_lcoohor( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->coohor = (coohor_s *)calloc( mem, sizeof(coohor_s) )) = = NULL ) {
                printf("\n Fallo al reservar %d coohor_s *", mem ) ;
                exit(1) ;
                }
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->coohor = (coohor_s *)realloc( lis->coohor, mem * sizeof(coohor_s) )) = = NULL ) {
                printf("\n Fallo al realocar %d coohor_s *", mem ) ;
                exit(1) ;
                }
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------*/
/*  mem_lcoover  Gesti¢n de memoria de una lista de coordenadas verticales  */
/*-------------------------------------------------------------------*/

lcoover_s mem_lcoover( int  mem,             /* n£mero de elementos */
                       lcoover_s *lis )              /* lista de coordenadas verticales */
{
```

```
if( mem < = 0 ) {
        *lis = fre_lcoover( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->coover = (coover_s *)calloc( mem, sizeof(coover_s) )) = = NULL ) {
                printf("\n Fallo al reservar %d coover_s *", mem ) ;
                exit(1) ;
                }
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->coover = (coover_s *)realloc( lis->coover, mem * sizeof(coover_s) )) = = NULL ) {
                printf("\n Fallo al realocar %d coover_s *", mem ) ;
                exit(1) ;
                }
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------------*/
/*   mem_lflecha    Gesti¢n de memoria de una lista de flechas       */
/*-------------------------------------------------------------------------*/

lflecha_s mem_lflecha( int          mem,    /* n£mero de elementos */
                       lflecha_s    *lis )  /* lista de flechas */
{

if( mem < = 0 ) {
        *lis = fre_lflecha( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->flecha = (flecha_s *)calloc( mem, sizeof(flecha_s) )) = = NULL ) {
                printf("\n Fallo al reservar %d flecha_s", mem ) ;
                exit(1) ;
                }
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->flecha = (flecha_s *)realloc( lis->flecha, mem * sizeof(flecha_s) )) = = NULL ) {
                printf("\n Fallo al realocar %d flecha_s", mem ) ;
                exit(1) ;
                }
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------------*/
```

```
/*   mem_lfledat    Gesti¢n de memoria de una lista de flechas dato              */
/*-------------------------------------------------------------------------*/

lfledat_s mem_lfledat( int              mem,    /* n£mero de elementos */
                       lfledat_s        *lis )  /* lista de flechas dato */
{

if( mem < = 0 ) {
        *lis = fre_lfledat( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->fledat = (fledat_s *)calloc( mem, sizeof(fledat_s) )) = = NULL ) {
                printf("\n Fallo al reservar %d fledat_s", mem ) ;
                exit(1) ;
                }
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->fledat = (fledat_s *)realloc( lis->fledat, mem * sizeof(fledat_s) )) = = NULL ) {
                printf("\n Fallo al realocar %d fledat_s", mem ) ;
                exit(1) ;
                }
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------------*/
/*   mem_lfunhor  Gesti¢n de memoria de una lista de funciones horizontales */
/*-------------------------------------------------------------------------*/

lfunhor_s mem_lfunhor( int   mem,               /* n£mero de elementos */
                       lfunhor_s *lis )                 /* lista de funciones horizontales */
{

if( mem < = 0 ) {
        *lis = fre_lfunhor( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->funhor = (funhor_s *)calloc( mem, sizeof(funhor_s) )) = = NULL ) {
                printf("\n Fallo al reservar %d funhor_s", mem ) ;
                exit(1) ;
                }
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->funhor = (funhor_s *)realloc( lis->funhor, mem * sizeof(funhor_s) )) = = NULL ) {
                printf("\n Fallo al realocar %d funhor_s", mem ) ;
                exit(1) ;
                }
```

```
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------*/
/*   mem_lfunver  Gesti¢n de memoria de una lista de funciones verticales   */
/*-------------------------------------------------------------------*/

lfunver_s mem_lfunver( int    mem,              /* n£mero de elementos */
                       lfunver_s *lis )                /* lista de funciones verticales */
{

if( mem < = 0 ) {
        *lis = fre_lfunver( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->funver = (funver_s *)calloc( mem, sizeof(funver_s) )) = = NULL ) {
                printf("\n Fallo al reservar %d funver_s", mem ) ;
                exit(1) ;
                }
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->funver = (funver_s *)realloc( lis->funver, mem * sizeof(funver_s) )) = = NULL ) {
                printf("\n Fallo al realocar %d funver_s", mem ) ;
                exit(1) ;
                }
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------*/
/*   mem_lgrupo    Gesti¢n de memoria de una lista de grupos              */
/*-------------------------------------------------------------------*/

lgrupo_s mem_lgrupo( int    mem,              /* n£mero de elementos */
                     lgrupo_s       *lis )          /* lista de grupos */
{
                     int    i ;                /* Contador de bucle */

if( mem < = 0 ) {
        *lis = fre_lgrupo( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->grupo = (grupo_s *)calloc( mem, sizeof(grupo_s) )) = = NULL ) {
                printf("\n Fallo al reservar %d grupo_s", mem ) ;
                exit(1) ;
                }
        for( i=0 ; i<mem ; i++ )
```

```
                lis->grupo[i].conjunto.relele.text.tt.s = NULL ;
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->grupo = (grupo_s *)realloc( lis->grupo, mem * sizeof(grupo_s) )) == NULL ) {
                printf("\n Fallo al realocar %d grupo_s", mem ) ;
                exit(1) ;
                }
        for( i=lis->m ; i<mem ; i++ )
                lis->grupo[i].conjunto.relele.text.tt.s = NULL ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------*/
/*   mem_lhueh  Gesti¢n de memoria de una lista de huecos horizontales   */
/*-------------------------------------------------------------------*/

lhueh_s mem_lhueh( int          mem,                    /* n£mero de elementos */
                   lhueh_s *lis )               /* lista de huecos horizontales */
{

if( mem <= 0 ) {
        *lis = fre_lhueh( lis ) ; return( *lis ) ;
        }
if( mem == lis->m )
        return( *lis ) ;

if( lis->m == 0 ) {
        if( (lis->hueh = (hueh_s *)calloc( mem, sizeof(hueh_s) )) == NULL ) {
                printf("\n Fallo al reservar %d hueh_s", mem ) ;
                exit(1) ;
                }
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->hueh = (hueh_s *)realloc( lis->hueh, mem * sizeof(hueh_s) )) == NULL ) {
                printf("\n Fallo al realocar %d hueh_s", mem ) ;
                exit(1) ;
                }
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------*/
/*   mem_lhuev  Gesti¢n de memoria de una lista de huecos verticales   */
/*-------------------------------------------------------------------*/

lhuev_s mem_lhuev( int          mem,                    /* n£mero de elementos */
                   lhuev_s *lis )               /* lista de huecos verticales */
{

if( mem <= 0 ) {
```

```
        *lis = fre_lhuev( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->huev = (huev_s *)calloc( mem, sizeof(huev_s) )) = = NULL ) {
                printf("\n Fallo al reservar %d huev_s", mem ) ;
                exit(1) ;
                }
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->huev = (huev_s *)realloc( lis->huev, mem * sizeof(huev_s) )) = = NULL ) {
                printf("\n Fallo al realocar %d huev_s", mem ) ;
                exit(1) ;
                }
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------------*/
/*   mem_linterf    Gesti¢n de memoria de una lista de inter con funci¢n   */
/*-------------------------------------------------------------------------*/

linterf_s mem_linterf( int      mem,            /* n£mero de elementos */
                       linterf_s *lis )         /* lista de inter con funci¢n */
{
                       int   i ;                /* Contador de bucle */

if( mem < = 0 ) {
        *lis = fre_linterf( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->interf = (interf_s *)calloc( mem, sizeof(interf_s) )) = = NULL ) {
                printf("\n Fallo al reservar %d interf_s", mem ) ;
                exit(1) ;
                }
        for( i=0 ; i<mem ; i+ + )
                lis->interf[i].fun = NULL ;
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->interf = (interf_s *)realloc( lis->interf, mem * sizeof(interf_s) )) = = NULL ) {
                printf("\n Fallo al realocar %d interf_s", mem ) ;
                exit(1) ;
                }
        for( i=lis->m ; i<mem ; i+ + )
                lis->interf[i].fun = NULL ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
```

```
}

/*-------------------------------------------------------------------------*/
/*   mem_liof    Gesti¢n de memoria de una lista de bloques if          */
/*-------------------------------------------------------------------------*/

liof_s mem_liof( int            mem,            /* n£mero de elementos */
                 liof_s         *lis )          /* lista de inpouts con funci¢n */
{
                 int            i ;             /* Contador de bucle */

if( mem < = 0 ) {
        *lis = fre_liof( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->iof = (iof_s *)calloc( mem, sizeof(iof_s) )) = = NULL ) {
                printf("\n Fallo al reservar %d iof_s", mem ) ;
                exit(1) ;
                }
        for( i=0 ; i<mem ; i+ + ) {
                lis->iof[i].fun        = NULL ;
                lis->iof[i].inpout.nom = NULL ;
                }
        lis->n = 0 ;
        lis->m = mem ;          /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->iof = (iof_s *)realloc( lis->iof, mem * sizeof(iof_s) )) = = NULL ) {
                printf("\n Fallo al realocar %d iof_s", mem ) ;
                exit(1) ;
                }
        for( i=lis->m ; i<mem ; i+ + ) {
                lis->iof[i].fun        = NULL ;
                lis->iof[i].inpout.nom = NULL ;
                }
        lis->m = mem ;          /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------------*/
/*   mem_liop    Gesti¢n de memoria de una lista de punteros inpout        */
/*-------------------------------------------------------------------------*/

liop_s mem_liop( int            mem,            /* n£mero de elementos */
                 liop_s         *lis )          /* lista de punteros inpouts */
{

if( mem < = 0 ) {
        *lis = fre_liop( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
```

```
        if( (lis->inpout = (inpout_s **)calloc( mem, sizeof(inpout_s *) )) == NULL ) {
                printf("\n Fallo al reservar %d inpout_s *", mem ) ;
                exit(1) ;
                }
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->inpout = (inpout_s **)realloc( lis->inpout, mem * sizeof(inpout_s *) )) == NULL ) {
                printf("\n Fallo al realocar %d inpout_s *", mem ) ;
                exit(1) ;
                }
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------*/
/*      mem_llindo          Gesti¢n de memoria de una lista de l¡neas dobles    */
/*-------------------------------------------------------------------*/

llindo_s mem_llindo( int     mem,               /* n£mero de elementos */
                     llindo_s *lis )            /* lista de l¡neas dobles */
{

if( mem <= 0 ) {
        *lis = fre_llindo( lis ) ; return( *lis ) ;
        }
if( mem == lis->m )
        return( *lis ) ;

if( lis->m == 0 ) {
        if( (lis->lindo = (lindo_s *)calloc( mem, sizeof(lindo_s) )) == NULL ) {
                printf("\n Fallo al reservar %d lindo_s *", mem ) ;
                exit(1) ;
                }
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->lindo = (lindo_s *)realloc( lis->lindo, mem * sizeof(lindo_s) )) == NULL ) {
                printf("\n Fallo al realocar %d lindo_s *", mem ) ;
                exit(1) ;
                }
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------*/
/*      mem_llinea          Gesti¢n de memoria de una lista de l¡neas          */
/*-------------------------------------------------------------------*/

llinea_s mem_llinea( int     mem,               /* n£mero de elementos */
                     llinea_s *lis )            /* lista de l¡neas */
{
```

```
if( mem < = 0 ) {
        *lis = fre_llinea( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->linea = (linea_s *)calloc( mem, sizeof(linea_s) )) = = NULL ) {
                printf("\n Fallo al reservar %d linea_s *", mem ) ;
                exit(1) ;
                }
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->linea = (linea_s *)realloc( lis->linea, mem * sizeof(linea_s) )) = = NULL ) {
                printf("\n Fallo al realocar %d linea_s *", mem ) ;
                exit(1) ;
                }
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------*/
/*   mem_lmarca    Gesti¢n de memoria de una lista de marcas          */
/*-------------------------------------------------------------------*/

lmarca_s mem_lmarca( int   mem,              /* n£mero de elementos */
                     lmarca_s      *lis )          /* lista de marcas */
{

if( mem < = 0 ) {
        *lis = fre_lmarca( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->marca = (marca_s *)calloc( mem, sizeof(marca_s) )) = = NULL ) {
                printf("\n Fallo al reservar %d marca_s", mem ) ;
                exit(1) ;
                }
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->marca = (marca_s *)realloc( lis->marca, mem * sizeof(marca_s) )) = = NULL ) {
                printf("\n Fallo al realocar %d marca_s", mem ) ;
                exit(1) ;
                }
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------*/
```

```
/*   mem_lrect    Gesti¢n de memoria de una lista de rectangulos     */
/*-------------------------------------------------------------------*/

lrect_s mem_lrect( int                  mem,            /* n£mero de elementos */
                   lrect_s *lis )               /* lista de rectangulos */
{

if( mem < = 0 ) {
        *lis = fre_lrect( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->rect = (rect_s *)calloc( mem, sizeof(rect_s) )) = = NULL ) {
                printf("\n Fallo al reservar %d rect_s *", mem ) ;
                exit(1) ;
                }
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->rect = (rect_s *)realloc( lis->rect, mem * sizeof(rect_s) )) = = NULL ) {
                printf("\n Fallo al realocar %d rect_s *", mem ) ;
                exit(1) ;
                }
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }

}

/*-------------------------------------------------------------------*/
/*   mem_ltexele    Gesti¢n de memoria de una lista de textos                */
/*-------------------------------------------------------------------*/

ltexele_s mem_ltexele( int    mem,              /* n£mero de elementos */
                       ltexele_s *lis )                 /* lista de textos */
{
                       int   i ;                /* Contador de bucle */

if( mem < = 0 ) {
        *lis = fre_ltexele( lis ) ; return( *lis ) ;
        }
if( mem = = lis->m )
        return( *lis ) ;

if( lis->m = = 0 ) {
        if( (lis->texele = (texele_s *)calloc( mem, sizeof(texele_s) )) = = NULL ) {
                printf("\n Fallo al reservar %d texele_s", mem ) ;
                exit(1) ;
                }
        for( i=0 ; i<mem ; i++ )
                lis->texele[i].s = NULL ;
        lis->n = 0 ;
        lis->m = mem ;              /*imp_mem() ;*/
        return( *lis ) ;
        }
else    {
        if( (lis->texele = (texele_s *)realloc( lis->texele, mem * sizeof(texele_s) )) = = NULL ) {
```

```
                printf("\n Fallo al realocar %d texele_s", mem ) ;
                exit(1) ;
                }
        for( i=lis->m ; i<mem ; i++ )
                lis->texele[i].s = NULL ;
        lis->m = mem ;                  /*imp_mem() ;*/
        return( *lis ) ;
        }

}
```

```
/*-------------------------------------------------------------------*/
/*        mem_dibu.fun      Prototipos de las funciones en mem_dibu.c      */
/*-------------------------------------------------------------------*/

/*-------------------------------------------------------------------*/
/*   fre_conjunto   Liberaci¢n de memoria de una estructura conjunto   */
conjunto_s fre_conjunto( conjunto_s   * ) ;      /* Una estructura conjunto */
/*-------------------------------------------------------------------*/
/*   fre_dibu                  Liberaci¢n de memoria de un dibujo              */
dibujo_s fre_dibu( dibujo_s   * ) ;              /* Un bloque if */
/*-------------------------------------------------------------------*/
/*   fre_dimfun  Liberaci¢n de memoria de dimensi¢n de funciones           */
dimfun_s fre_dimfun( dimfun_s        * ) ;       /* Una estructura de dimensi¢n de funciones */
/*-------------------------------------------------------------------*/
/*   fre_dimhue  Liberaci¢n de memoria de dimensi¢n de huecos              */
dimhue_s fre_dimhue( dimhue_s        * ) ;       /* Una estructura de dimensi¢n de huecos */
/*-------------------------------------------------------------------*/
/*   fre_flecha      Liberaci¢n de memoria de una flecha                   */
flecha_s fre_flecha( flecha_s * ) ;              /* Una flecha */
/*-------------------------------------------------------------------*/
/*   fre_fledat      Liberaci¢n de memoria de una flecha dato          */
fledat_s fre_fledat( fledat_s   * ) ;            /* Una flecha dato */
/*-------------------------------------------------------------------*/
/*   fre_grupo       Liberaci¢n de memoria de un grupo                 */
grupo_s fre_grupo( grupo_s   * ) ;               /* Un grupo */
/*-------------------------------------------------------------------*/
/*   fre_inpout      Liberaci¢n de memoria de un inpout                */
inpout_s fre_inpout( inpout_s        * ) ;       /* Un inpout */
/*-------------------------------------------------------------------*/
/*   fre_inter       Liberaci¢n de memoria de una estructura inter     */
inter_s fre_inter( inter_s       * ) ;           /* Una estructura inter */
/*-------------------------------------------------------------------*/
/*   fre_interf  Liberaci¢n de memoria de una estructura inter con funci¢n  */
```

```
interf_s fre_interf( interf_s    * ) ;                  /* Una estructura interf */

/*-------------------------------------------------------------------------*/

/*   fre_iof          Liberaci¢n de memoria de un inpout con funci¢n                  */

iof_s fre_iof( iof_s  * ) ;                  /* Un inpout con funci¢n */
/*-------------------------------------------------------------------------*/

/*   fre_larco        Liberaci¢n de memoria de una lista de arcos        */

larco_s fre_larco( larco_s     * ) ;                  /* Lista de arcos */
/*-------------------------------------------------------------------------*/

/*   fre_lcoohor Liberaci¢n de memoria de una lista de coordenadas horizon. */

lcoohor_s fre_lcoohor( lcoohor_s * ) ;              /* Lista de arcos */
/*-------------------------------------------------------------------------*/

/*   fre_lcoover Liberaci¢n de memoria de una lista de coordenadas vertica. */

lcoover_s fre_lcoover( lcoover_s * ) ;              /* Lista de coordenadas verticales */
/*-------------------------------------------------------------------------*/

/*   fre_lfledat  Liberaci¢n de memoria de una lista de flechas dato                */

lfledat_s fre_lfledat( lfledat_s          * ) ;       /* Lista de flechas dato */
/*-------------------------------------------------------------------------*/

/*   fre_lflecha  Liberaci¢n de memoria de una lista de flechas          */

lflecha_s fre_lflecha( lflecha_s          * ) ;       /* Lista de flechas  */
/*-------------------------------------------------------------------------*/

/*  fre_lfunhor Liberaci¢n de memoria de una lista de funciones horizontales */

lfunhor_s fre_lfunhor( lfunhor_s * ) ;              /* Lista de funciones horizontales (dimensi¢n) */
/*-------------------------------------------------------------------------*/

/*  fre_lfunver Liberaci¢n de memoria de una lista de funciones verticales   */

lfunver_s fre_lfunver( lfunver_s * ) ;              /* Lista de funciones verticales (dimensi¢n) */
/*-------------------------------------------------------------------------*/

/*   fre_lgrupo  Liberaci¢n de memoria de una lista de grupos                   */

lgrupo_s fre_lgrupo( lgrupo_s          * ) ;              /* Lista de grupos */
/*-------------------------------------------------------------------------*/

/*   fre_lhueh    Liberaci¢n de memoria de una lista de huecos horizontales */

lhueh_s fre_lhueh( lhueh_s * ) ;                  /* Lista de huecos horizontales (dimensi¢n) */
/*-------------------------------------------------------------------------*/

/*   fre_lhuev    Liberaci¢n de memoria de una lista de huecos verticales   */

lhuev_s fre_lhuev( lhuev_s * ) ;                  /* Lista de huecos verticales (dimensi¢n) */

/*-------------------------------------------------------------------------*/
```

```
/*  fre_linea       Liberaci¢n de memoria de una estructura linea     */

linea_s fre_linea( linea_s     * ) ;              /* Una estructura linea */

/*--------------------------------------------------------------------*/

/*  fre_linterf  Liberaci¢n de memoria de una lista de inter con funci¢n  */

linterf_s fre_linterf( linterf_s * ) ;     /* Lista de inter con funci¢n */
/*--------------------------------------------------------------------*/

/*  fre_liof  Liberaci¢n de memoria de una lista de inpouts con funci¢n  */

liof_s fre_liof( liof_s         * ) ;              /* Lista de inpouts con funci¢n */
/*--------------------------------------------------------------------*/

/*  fre_liop  Liberaci¢n de memoria de una lista de inpouts puntero          */

liop_s fre_liop( liop_s         * ) ;              /* Lista de inpouts puntero */
/*--------------------------------------------------------------------*/

/*  fre_llindo  Liberaci¢n de memoria de una lista de l¡neas dobles          */

llindo_s fre_llindo( llindo_s   * ) ;              /* Lista de l¡neas dobles */
/*--------------------------------------------------------------------*/

/*  fre_llinea  Liberaci¢n de memoria de una lista de l¡neas                 */

llinea_s fre_llinea( llinea_s   * ) ;              /* Lista de l¡neas */
/*--------------------------------------------------------------------*/

/*  fre_lmarca  Liberaci¢n de memoria de una lista de marca                  */

lmarca_s fre_lmarca( lmarca_s       * ) ;              /* Lista de marcas */
/*--------------------------------------------------------------------*/

/*  fre_lrect  Liberaci¢n de memoria de una lista de rectangulos     */

lrect_s fre_lrect( lrect_s      * ) ;              /* Lista de rectangulos */
/*--------------------------------------------------------------------*/

/*  fre_ltexele    Liberaci¢n de memoria de una lista de textos      */

ltexele_s fre_ltexele( ltexele_s * ) ;            /* Lista de textos */

/*--------------------------------------------------------------------*/

/*  fre_relele      Liberaci¢n de memoria de una estructura relele    */

relele_s fre_relele( relele_s   * ) ;              /* Una estructura relele */

/*--------------------------------------------------------------------*/

/*  fre_texdib      Liberaci¢n de memoria de una estructura texdib           */

texdib_s fre_texdib( texdib_s* ) ;                 /* Una estructura texdib */

/*--------------------------------------------------------------------*/

/*  fre_texele      Liberaci¢n de memoria de un texto                        */
```

```
texele_s fre_texele( texele_s * ) ;                    /* Una estructura texele */

/*-------------------------------------------------------------------*/

/*   mem_larco    Gesti¢n de memoria de una lista de arcos              */

larco_s mem_larco( int                ,                /* n£mero de elementos */
                   larco_s * ) ;                       /* lista de arco */

/*-------------------------------------------------------------------*/

/*  mem_lcoohor  Gesti¢n de memoria de una lista de coordenadas horizontales */

lcoohor_s mem_lcoohor( int  ,                          /* n£mero de elementos */
                       lcoohor_s * ) ;                 /* lista de coordenadas horizontales */

/*-------------------------------------------------------------------*/

/*  mem_lcoover  Gesti¢n de memoria de una lista de coordenadas verticales  */

lcoover_s mem_lcoover( int  ,                          /* n£mero de elementos */
                       lcoover_s * ) ;                 /* lista de coordenadas verticales */

/*-------------------------------------------------------------------*/

/*   mem_lflecha    Gesti¢n de memoria de una lista de flechas        */

lflecha_s mem_lflecha( int            ,                /* n£mero de elementos */
                       lflecha_s      * ) ;            /* lista de flechas */

/*-------------------------------------------------------------------*/

/*   mem_lfledat    Gesti¢n de memoria de una lista de flechas dato   */

lfledat_s mem_lfledat( int            ,                /* n£mero de elementos */
                       lfledat_s      * ) ;            /* lista de flechas dato */

/*-------------------------------------------------------------------*/

/*   mem_lfunhor  Gesti¢n de memoria de una lista de funciones horizontales */

lfunhor_s mem_lfunhor( int  ,                          /* n£mero de elementos */
                       lfunhor_s * ) ;                 /* lista de funciones horizontales */

/*-------------------------------------------------------------------*/

/*   mem_lfunver  Gesti¢n de memoria de una lista de funciones verticales  */

lfunver_s mem_lfunver( int  ,                          /* n£mero de elementos */
                       lfunver_s * ) ;                 /* lista de funciones verticales */

/*-------------------------------------------------------------------*/

/*   mem_lgrupo    Gesti¢n de memoria de una lista de grupos          */

lgrupo_s mem_lgrupo( int    ,                          /* n£mero de elementos */
                     lgrupo_s      * ) ;                /* lista de grupos */

/*-------------------------------------------------------------------*/

/*   mem_lhueh  Gesti¢n de memoria de una lista de huecos horizontales  */
```

```
lhueh_s mem_lhueh( int        ,                    /* n£mero de elementos */
                   lhueh_s * ) ;          /* lista de huecos horizontales */

/*-------------------------------------------------------------------*/

/*   mem_lhuev  Gesti¢n de memoria de una lista de huecos verticales      */

lhuev_s mem_lhuev( int        ,                    /* n£mero de elementos */
                   lhuev_s * ) ;          /* lista de huecos verticales */

/*-------------------------------------------------------------------*/

/*   mem_linterf    Gesti¢n de memoria de una lista de inter con funci¢n  */

linterf_s mem_linterf( int      ,                  /* n£mero de elementos */
                       linterf_s * ) ;        /* lista de inter con funci¢n */

/*-------------------------------------------------------------------*/

/*   mem_liof    Gesti¢n de memoria de una lista de bloques if        */

liof_s mem_liof( int,                          /* n£mero de elementos */
                 liof_s    * ) ;             /* lista de inpouts con funci¢n */

/*-------------------------------------------------------------------*/

/*   mem_liop    Gesti¢n de memoria de una lista de punteros inpout        */

liop_s mem_liop( int                ,                /* n£mero de elementos */
                 liop_s             * ) ;            /* lista de punteros inpouts */

/*-------------------------------------------------------------------*/

/*        mem_llindo        Gesti¢n de memoria de una lista de l¡neas dobles    */

llindo_s mem_llindo( int    ,                  /* n£mero de elementos */
                     llindo_s * ) ;           /* lista de l¡neas dobles */

/*-------------------------------------------------------------------*/

/*        mem_llinea        Gesti¢n de memoria de una lista de l¡neas        */

llinea_s mem_llinea( int    ,                  /* n£mero de elementos */
                     llinea_s * ) ;           /* lista de l¡neas */

/*-------------------------------------------------------------------*/

/*   mem_lmarca    Gesti¢n de memoria de una lista de marcas              */

lmarca_s mem_lmarca( int  ,                    /* n£mero de elementos */
                     lmarca_s     * ) ;           /* lista de marcas */

/*-------------------------------------------------------------------*/

/*   mem_lrect    Gesti¢n de memoria de una lista de rectangulos     */

lrect_s mem_lrect( int              ,                /* n£mero de elementos */
                   lrect_s * ) ;            /* lista de rectangulos */

/*-------------------------------------------------------------------*/
```

```
/*   mem_ltexele    Gesti¢n de memoria de una lista de textos                    */

ltexele_s mem_ltexele( int     ,                  /* n£mero de elementos */
                       ltexele_s * ) ;            /* lista de textos */
```

DOCUMENTO N°2

PLANOS

En este documento se ha querido incluir una selección de los diagramas ANA realizados para diseñar la segunda versión (para la primera no se pudieron hacer). Consideramos que estos diagramas equivalen a los planos que se presentarían en un proyecto de otra rama de la ingeniería que no sea la Ingeniería del Conocimiento. Nos referiremos a los planos como funciones ya que cada plano corresponde a una función descrita.

1- Lista de planos

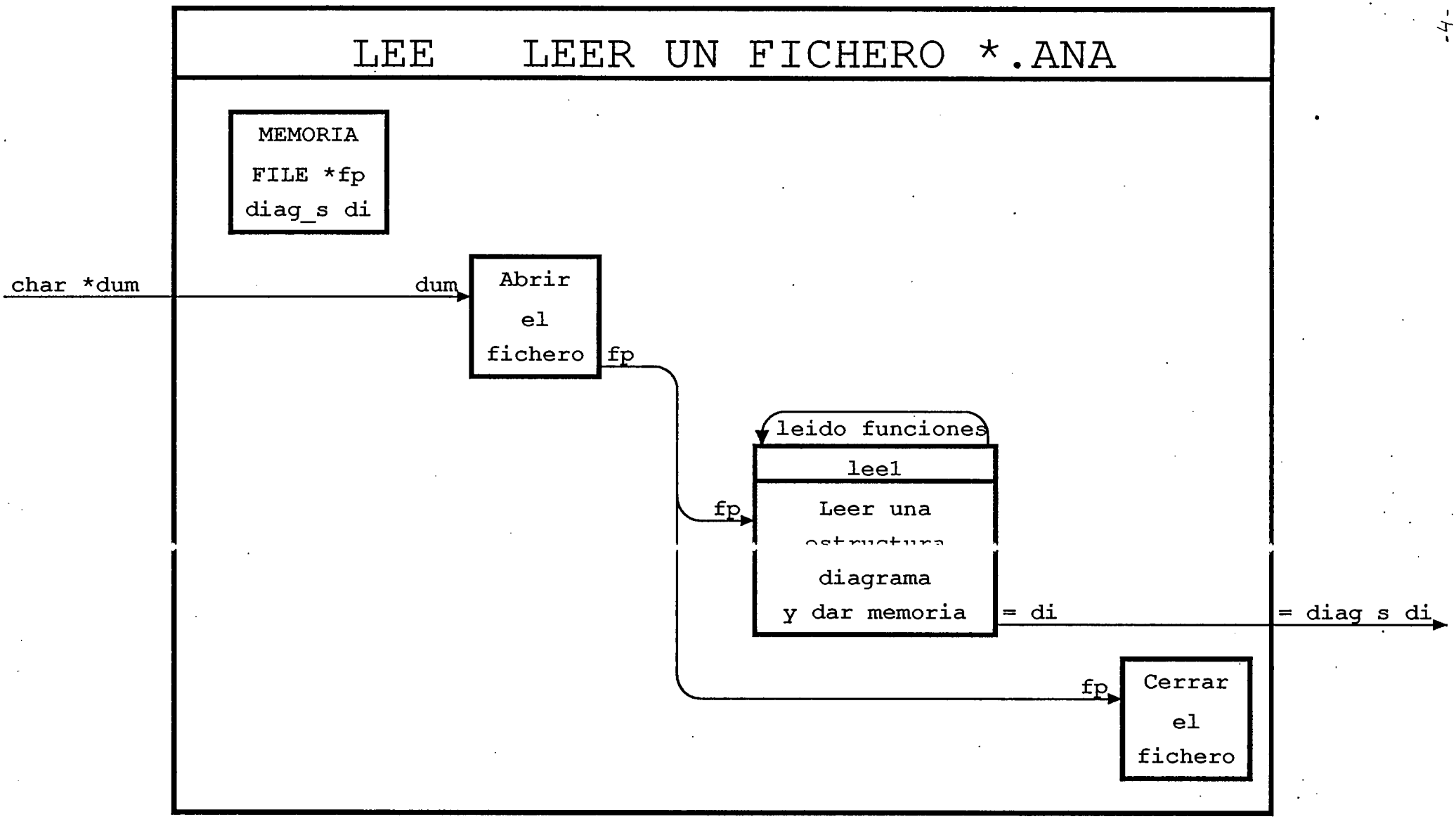
Primero tenemos descrito el árbol de la función que el fichero de extensión '.ANA' y lo carga en memoria en la estructura 'diagrama'. El nodo raíz es la función 'LEE'. Son los siguientes:

Luego tenemos el árbol de escritura de la estructura 'diagrama' a un fichero ASCII. El nodo raíz es la función 'ESC'. Son los siguientes:

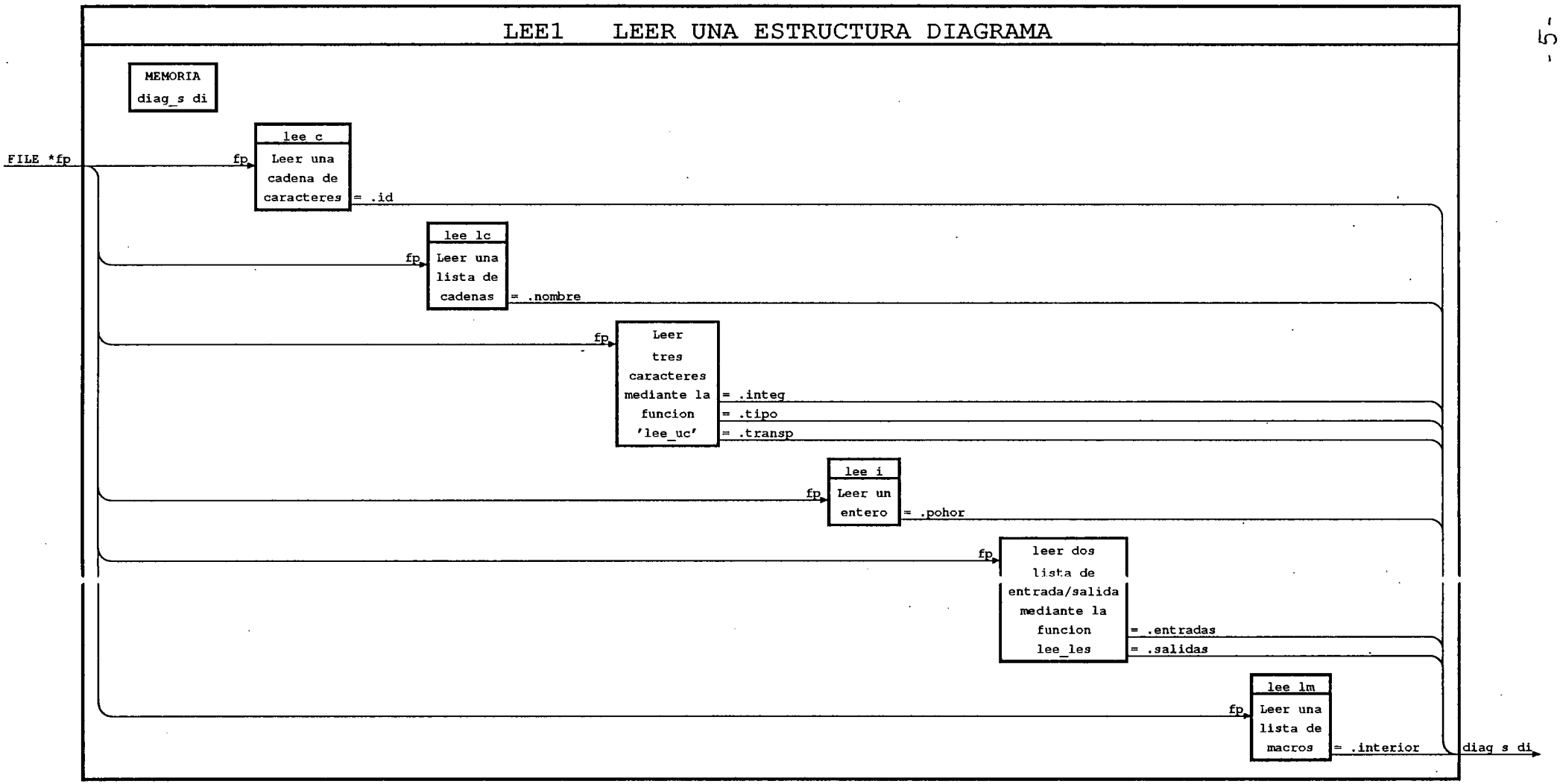
A continuación tenemos el árbol de la función que crea la estructura 'dibujo' a partir de la estructura 'diagrama'. Este árbol se ha dividido en dos para no tener una excesiva profundidad. El nodo raíz de la primera rama es 'DIB'. e¡El otro nodo raíz es 'CO' aunque dependa exclusivamente de la función 'DIB31'. A continuación listamos todas las funciones descritas:

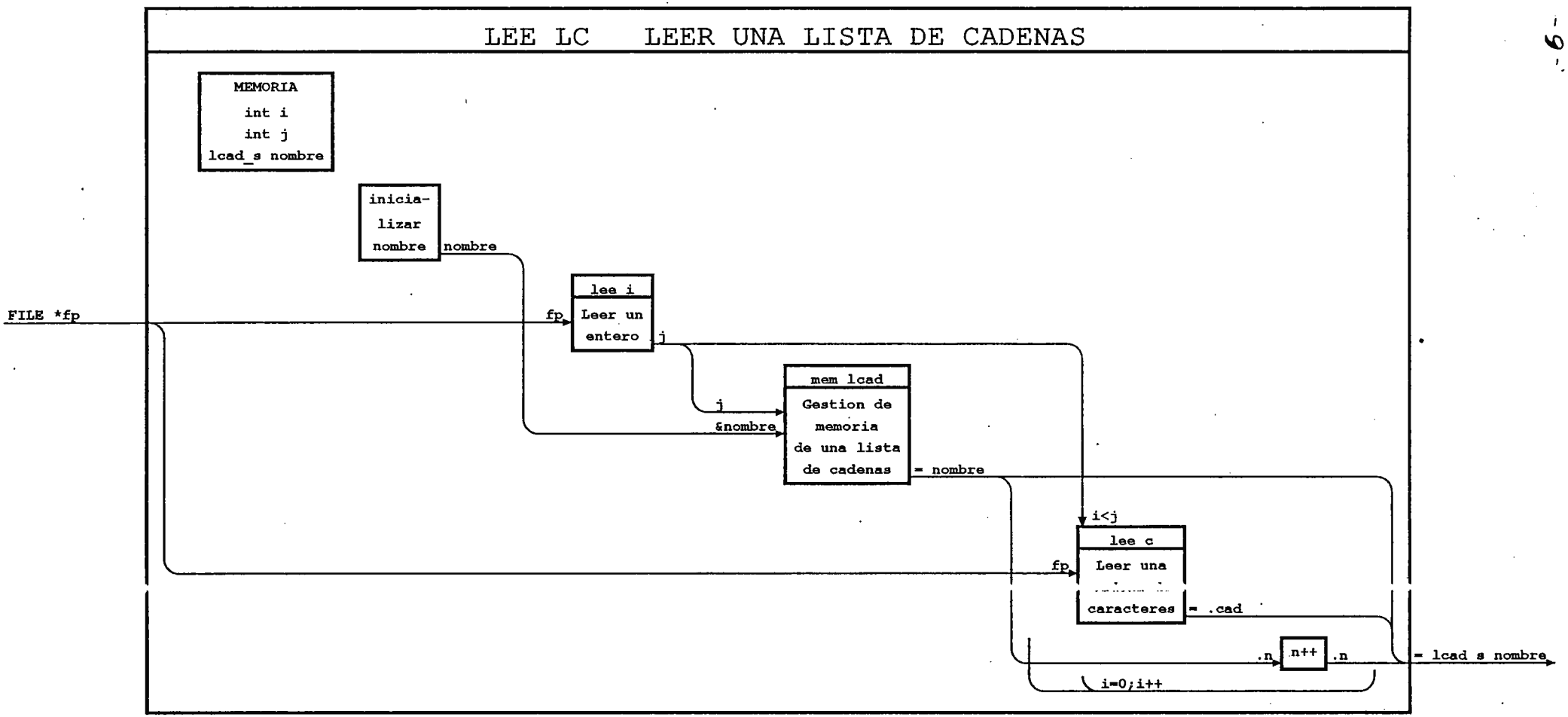
2- Planos

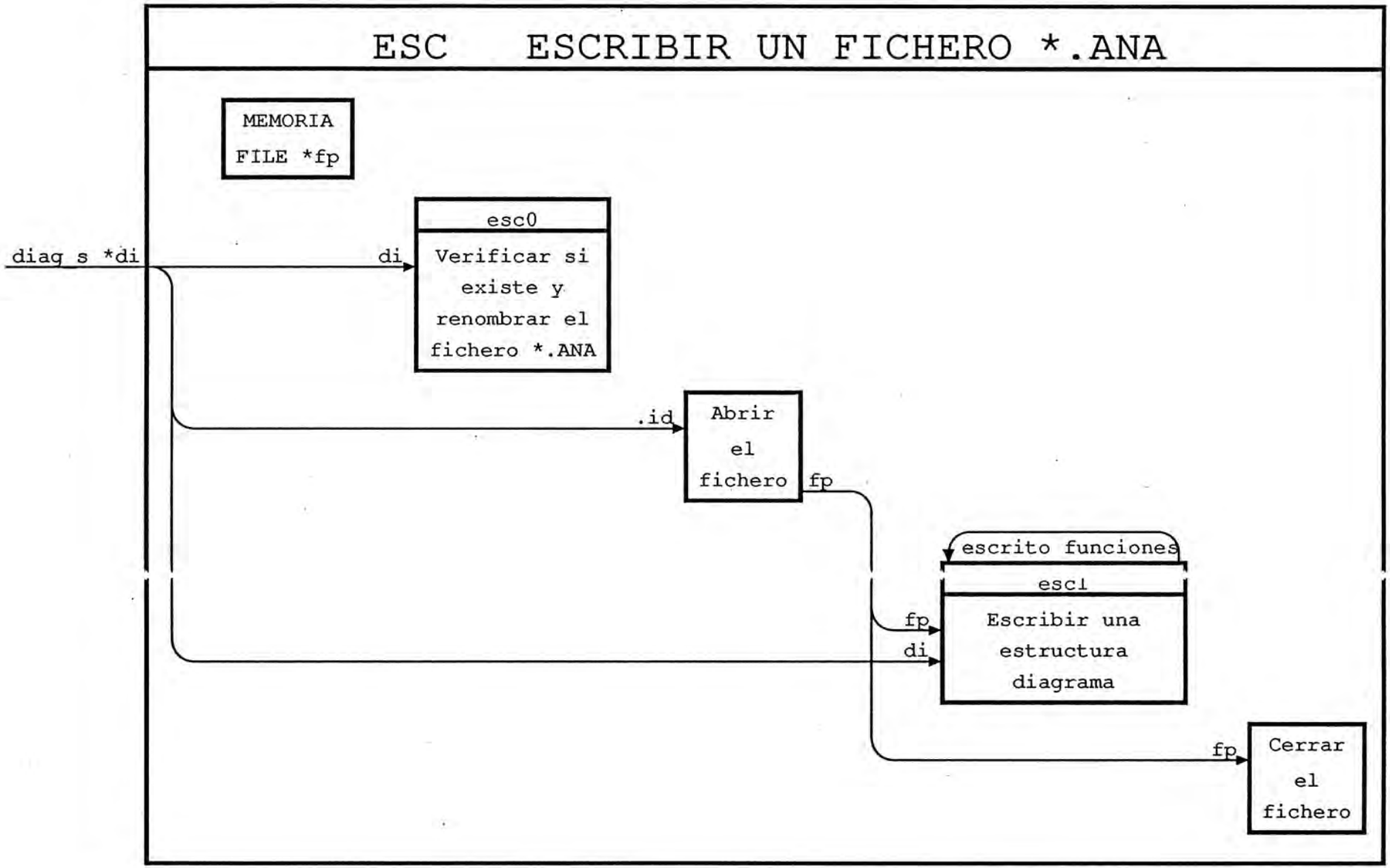
A continuación tenemos los planos en el orden listado en el apartado anterior.


LEE LEER UN FICHERO *.ANA
MEMORIA
FILE *fp
diag_s di
char *dum
dum
Abrir
el
fichero
fp
leido funciones
lee1
fp
Leer una
estructura
diagrama
y dar memoria
= di
= diag s di
fp
Cerrar
el
fichero

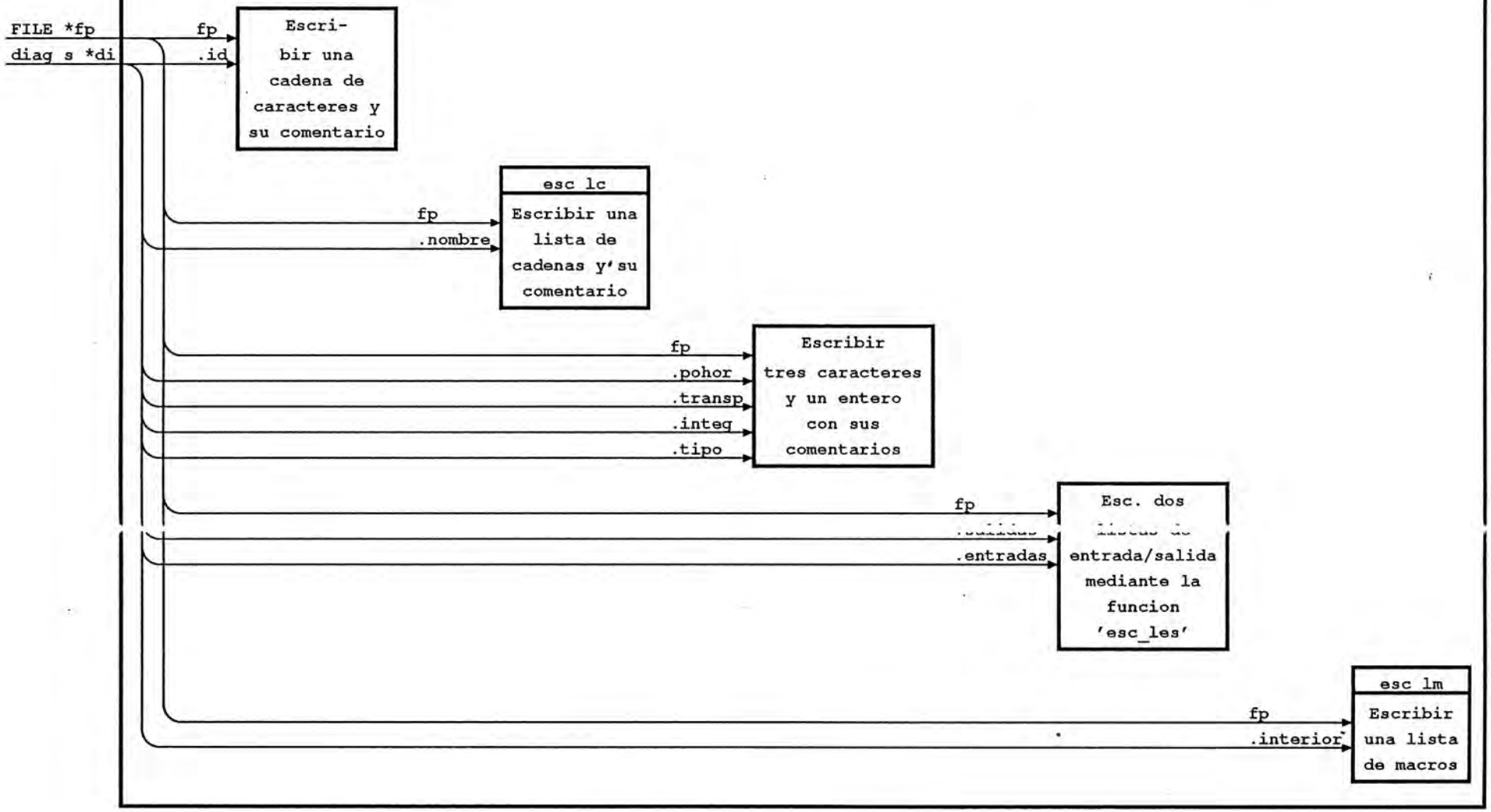
LEE1 LEER UNA ESTRUCTURA DIAGRAMA

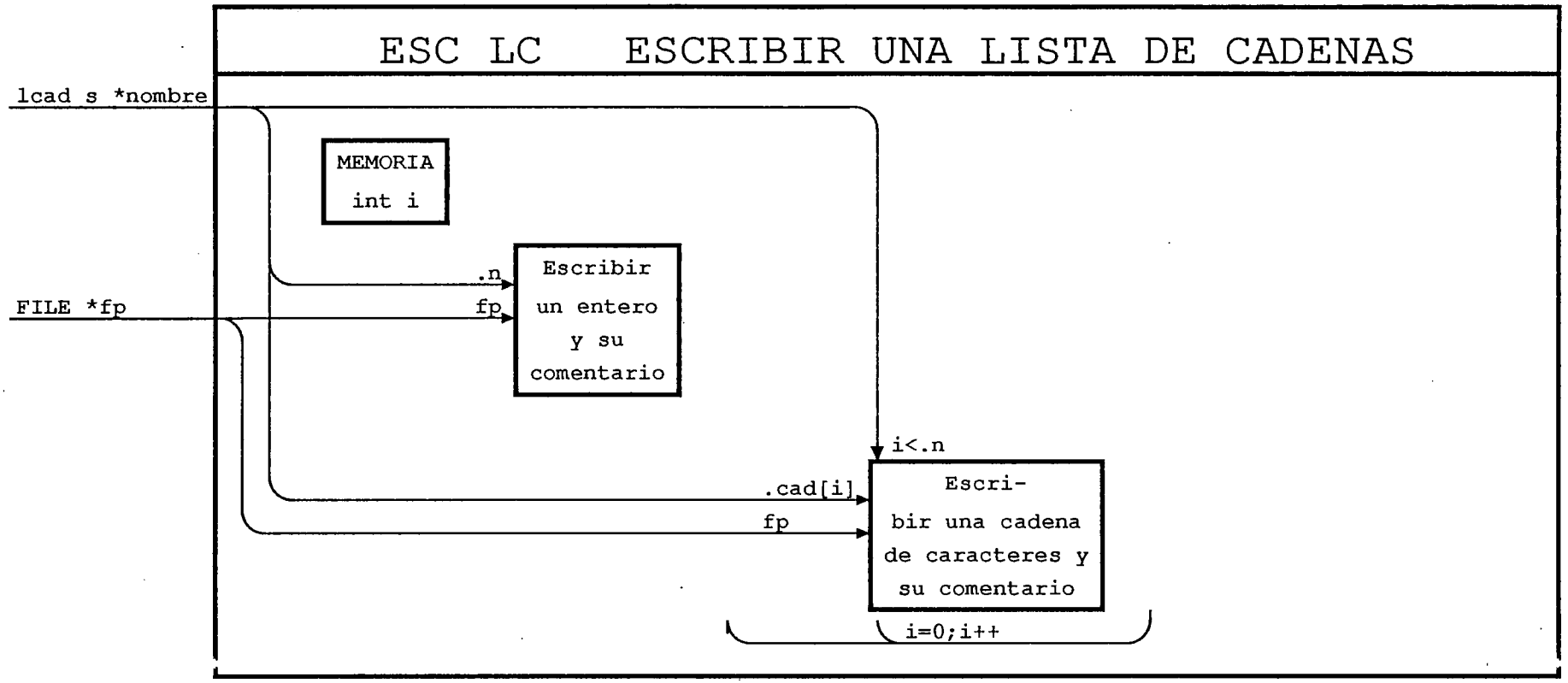


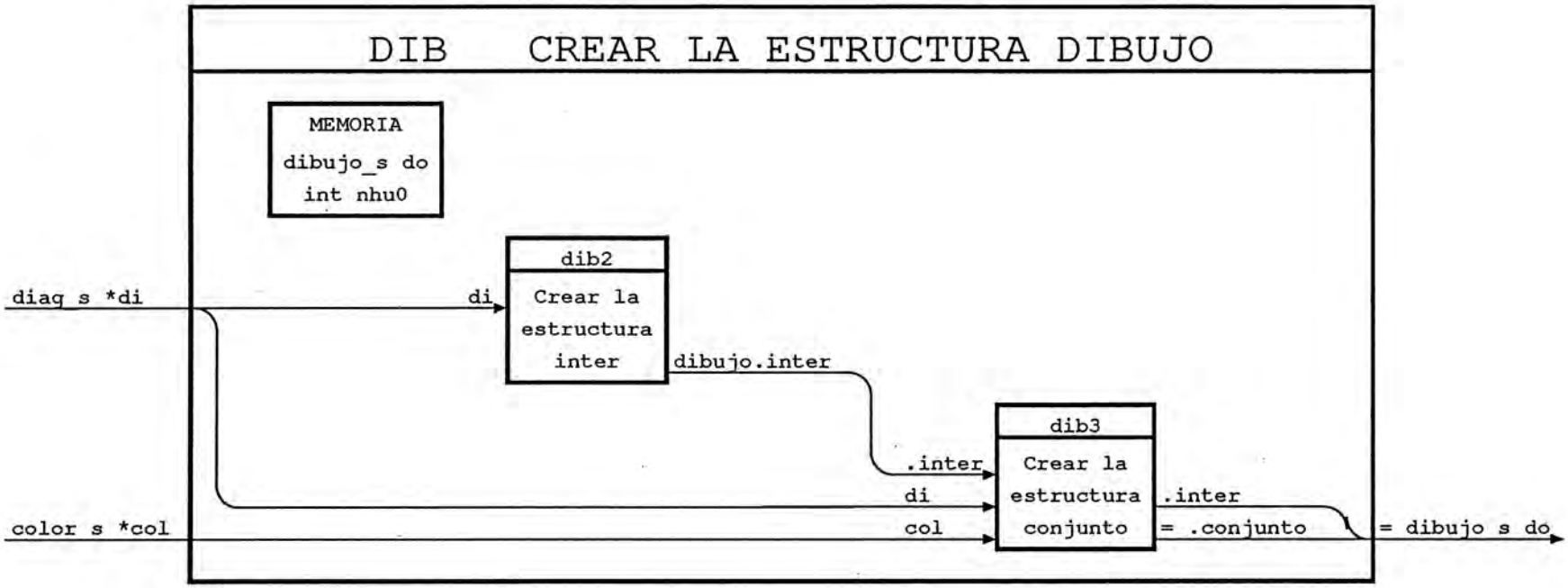

LEE LC LEER UNA LISTA DE CADENAS
MEMORIA
int i
int j
lcad_s nombre
inicia-
lizar
nombre
nombre
FILE *fp
fp
lee i
Leer un
entero
j
j
&nombre
mem lcad
Gestion de
memoria
de una lista
de cadenas
= nombre
i<j
lee c
Leer una
caracteres
fp
= .cad
.n
n++
.n
= lcad_s nombre
i=0;i++

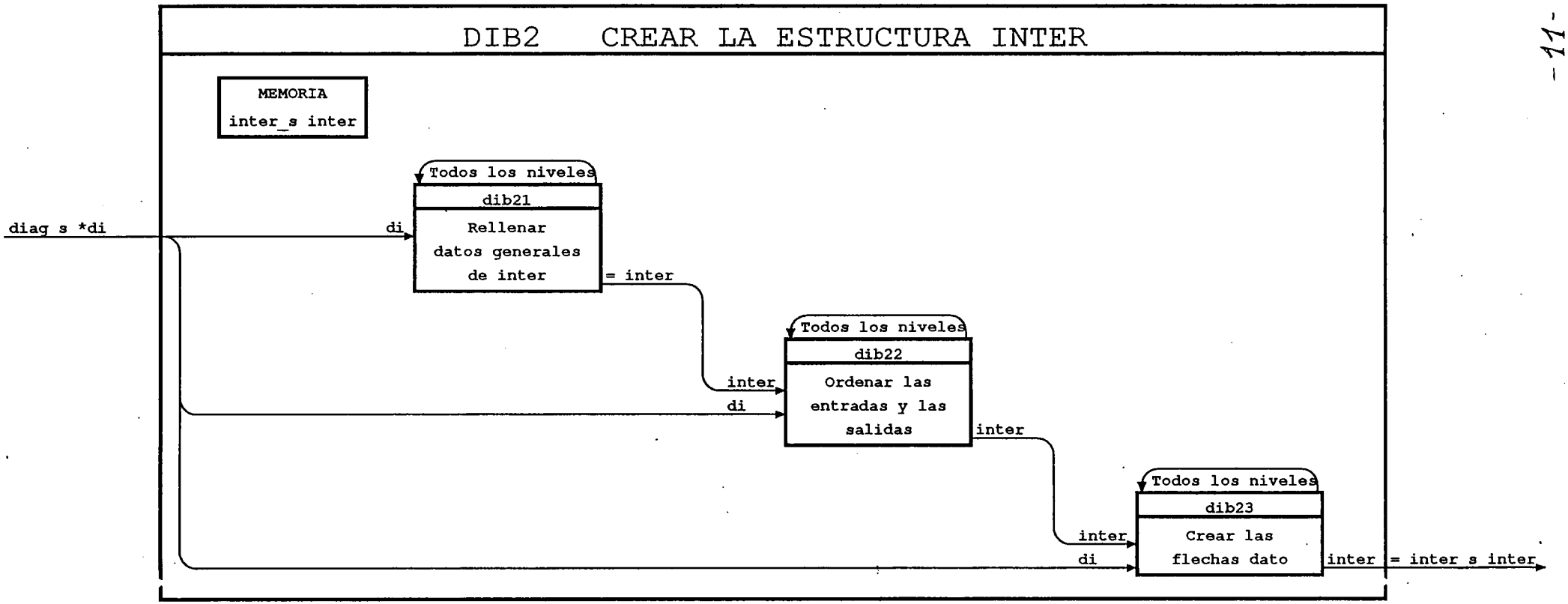

ESC ESCRIBIR UN FICHERO *.ANA
MEMORIA
FILE *fp
esc0
Verificar si existe y renombrar el fichero *.ANA
diag_s *di
di
.id
Abrir el fichero
fp
escrito funciones
esc1
Escribir una estructura diagrama
fp
di
fp
Cerrar el fichero

ESC1 ESCRIBIR UNA ESTRUCTURA DIAGRAMA

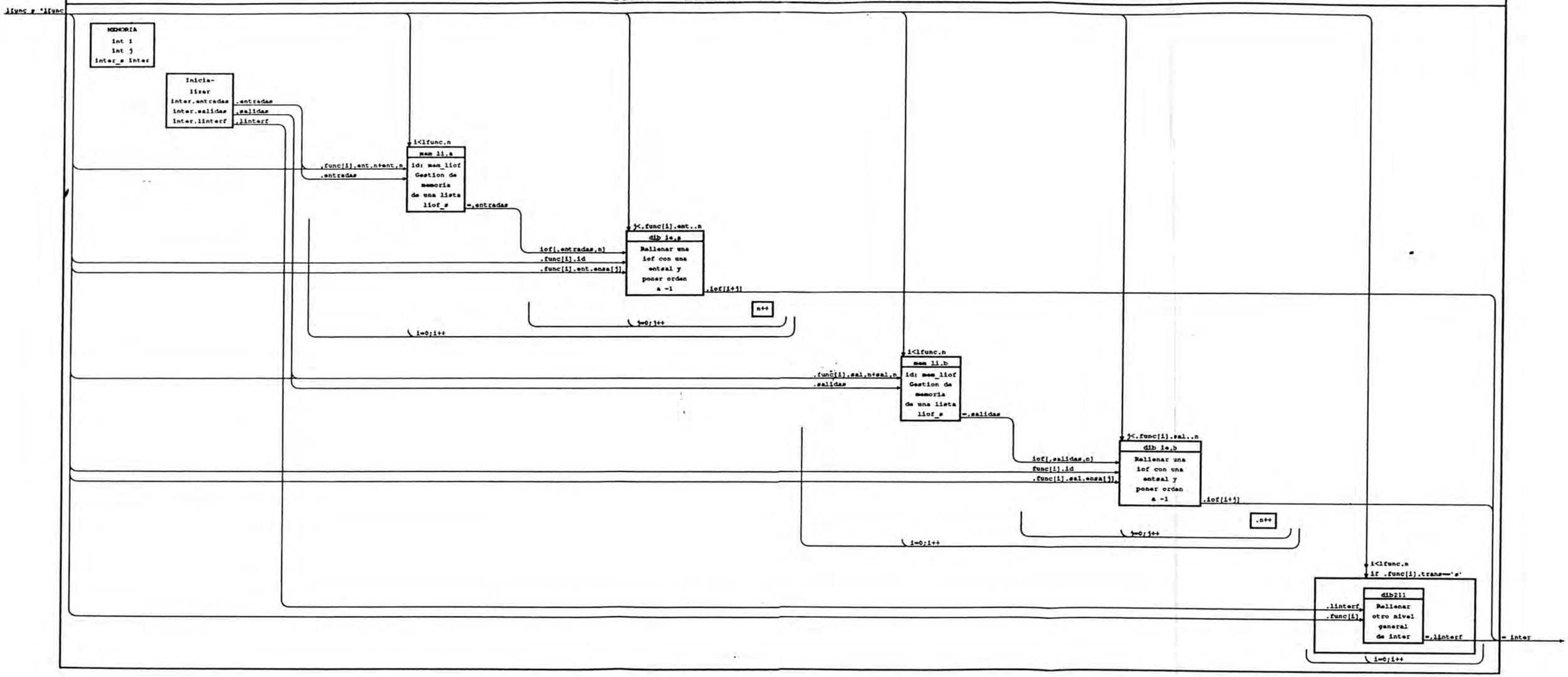


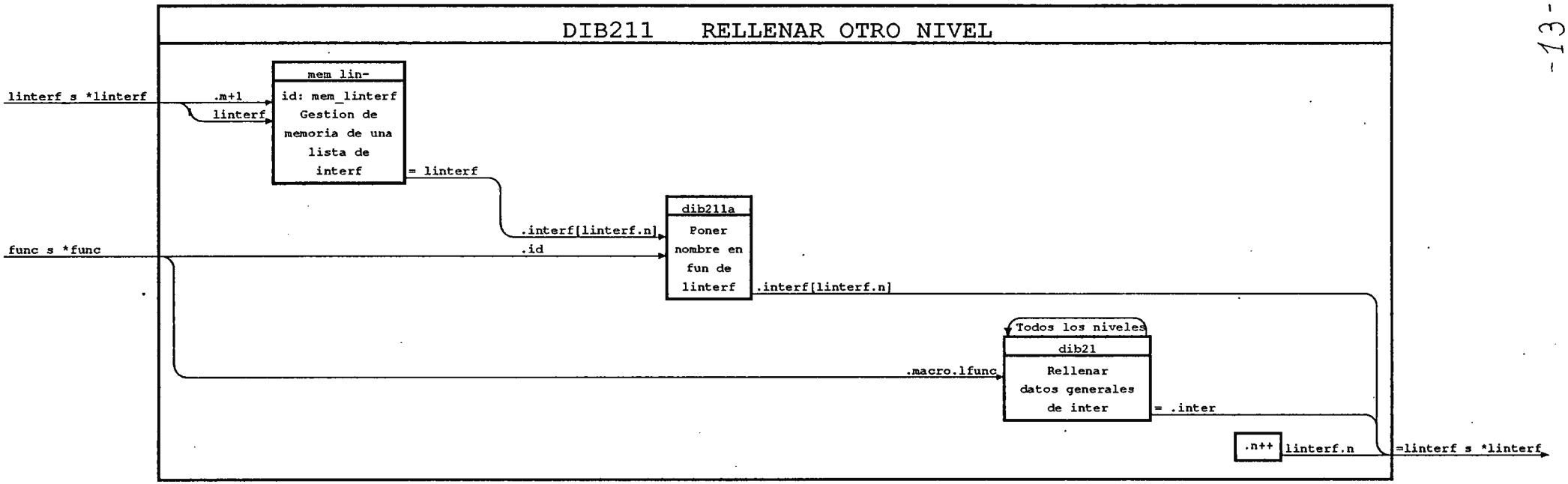

ESC LC ESCRIBIR UNA LISTA DE CADENAS
lcad s *nombre
MEMORIA
int i
.n
FILE *fp
fp
Escribir un entero y su comentario
i<.n
.cad[i]
fp
Escri-bir una cadena de caracteres y su comentario
i=0;i++

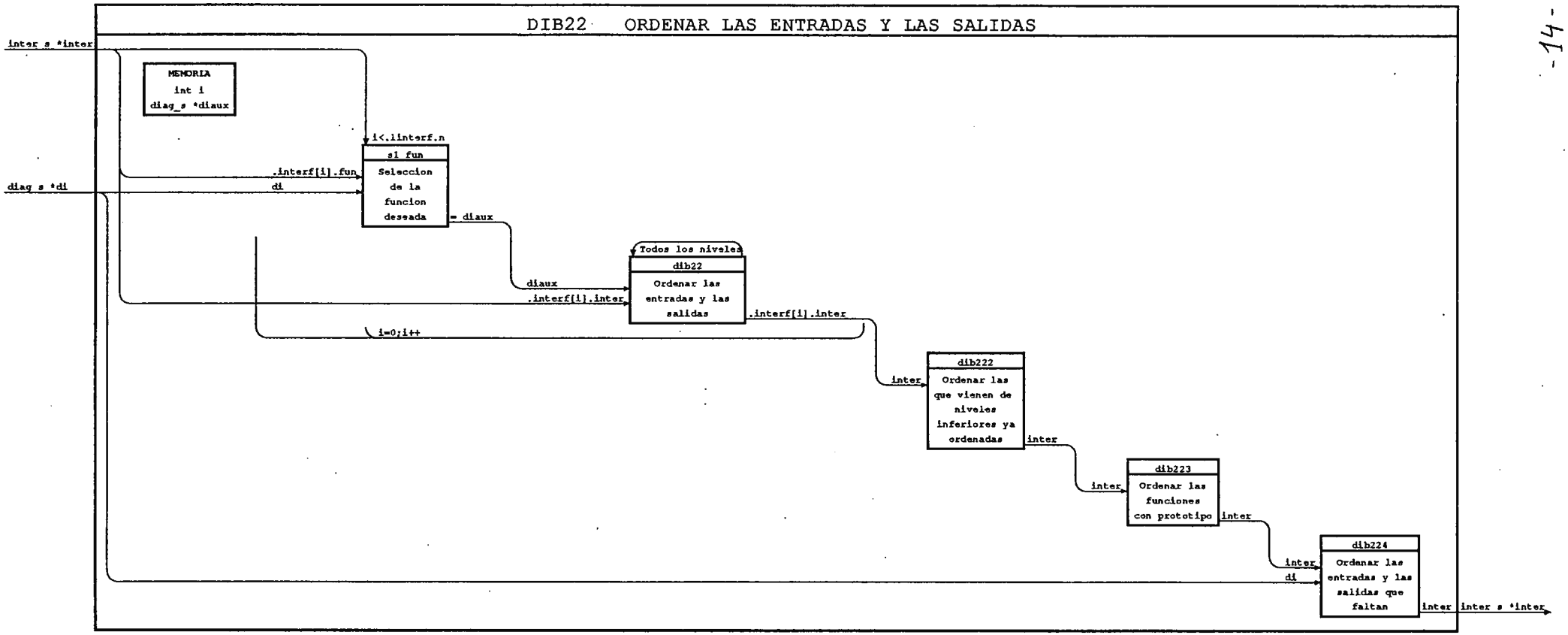

DIB CREAR LA ESTRUCTURA DIBUJO
MEMORIA
dibujo_s do
int nhu0
dib2
Crear la
estructura
inter
diag s *di
di
dibujo.inter
dib3
Crear la
estructura
conjunto
.inter
di
col
.inter
= .conjunto
color s *col
= dibujo s do


DIB2 CREAR LA ESTRUCTURA INTER
MEMORIA
inter_s inter
Todos los niveles
dib21
Rellenar
datos generales
de inter
diag s *di
di
= inter
Todos los niveles
dib22
Ordenar las
entradas y las
salidas
inter
di
inter
Todos los niveles
dib23
Crear las
flechas dato
inter
di
inter = inter s inter

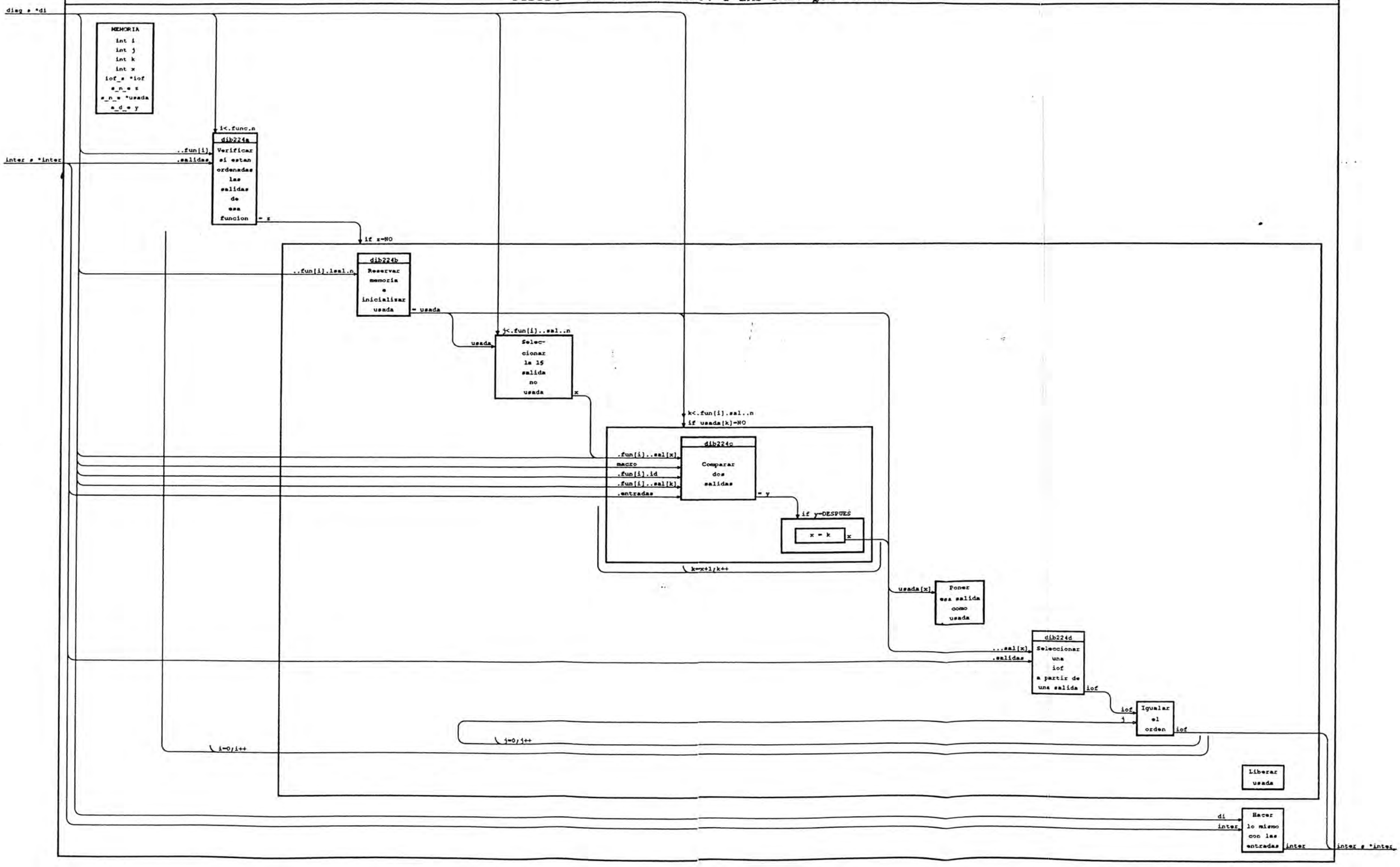
DIB21 RELLENAR DATOS GENERALES DE INTER

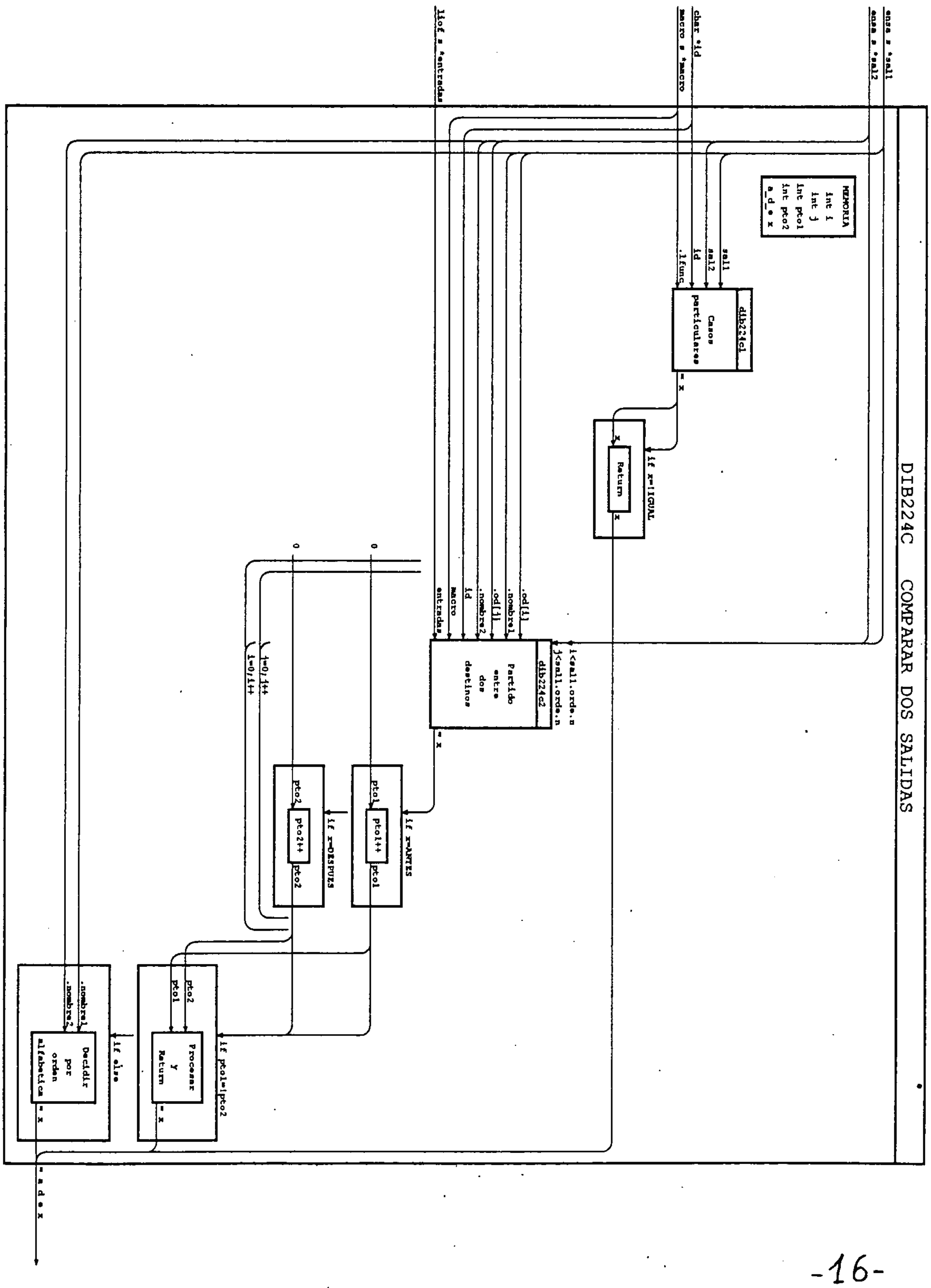


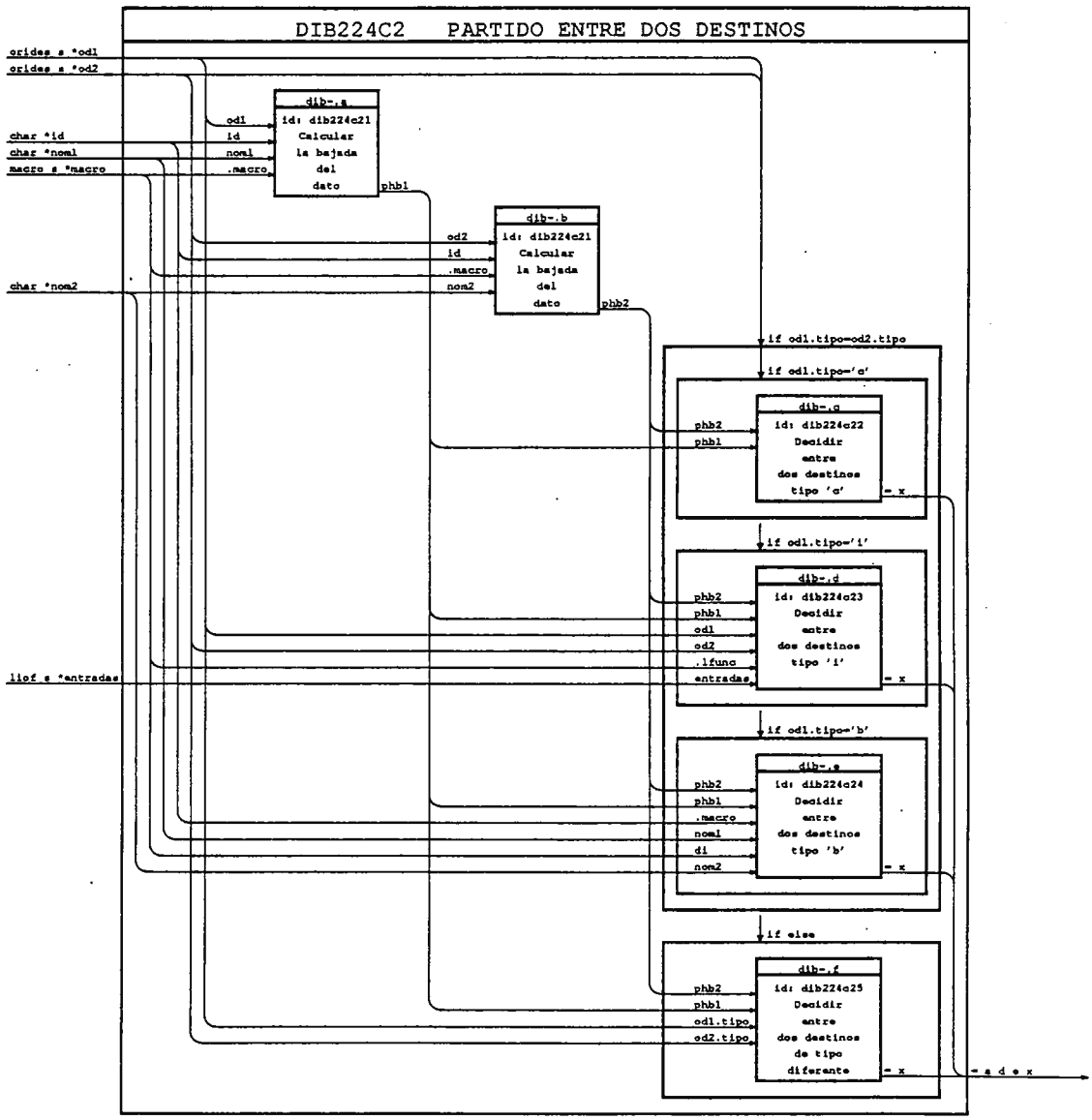

DIB211 RELLENAR OTRO NIVEL
linterf_s *linterf
.m+1
linterf
mem_lin-
id: mem_linterf
Gestion de memoria de una lista de interf
= linterf
.interf[linterf.n]
.id
dib211a
Poner nombre en fun de linterf
func s *func
.interf[linterf.n]
Todos los niveles
dib21
Rellenar datos generales de inter
.macro.lfunc
= .inter
.n++
linterf.n
=linterf s *linterf

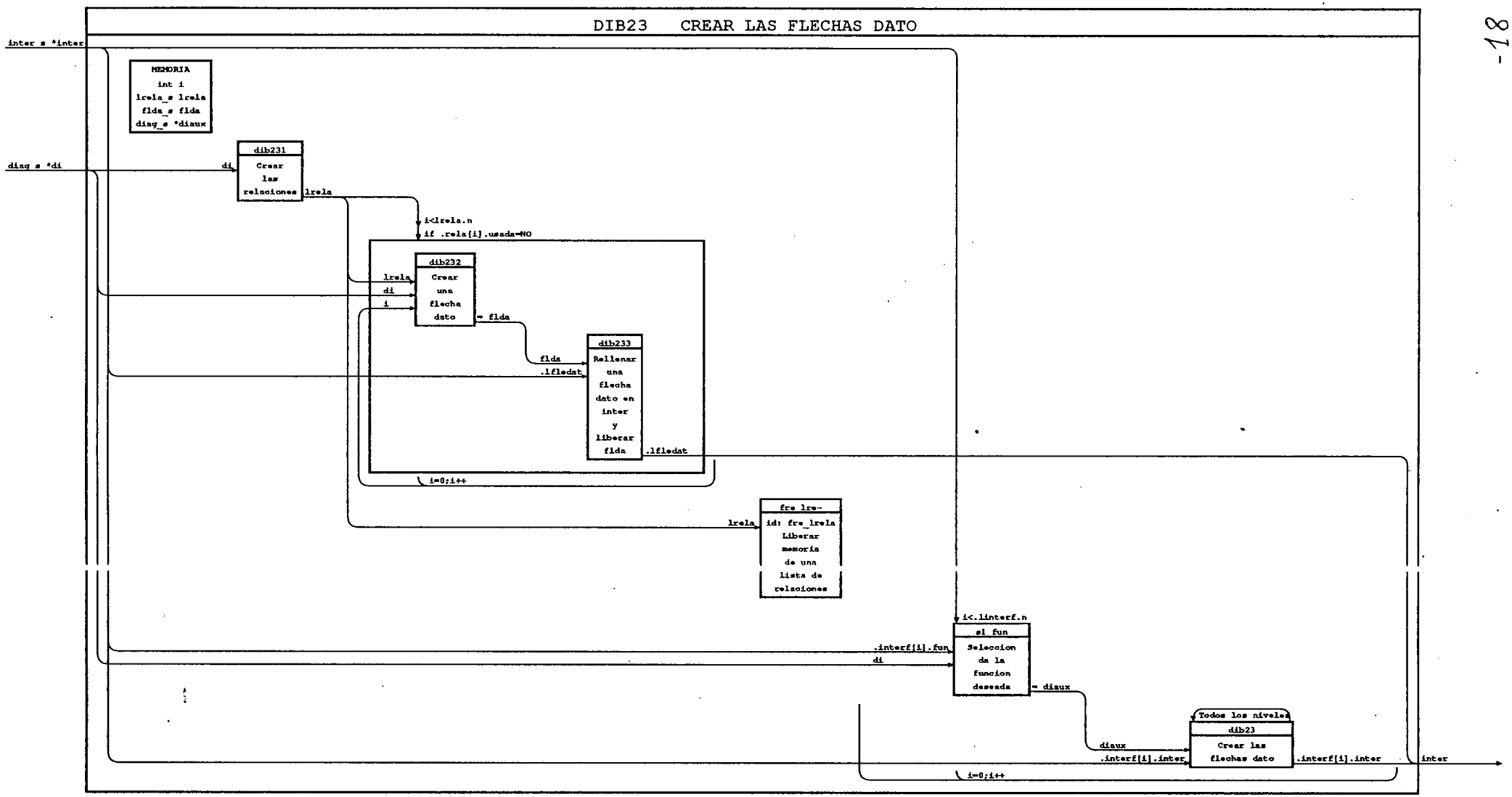

DIB22 ORDENAR LAS ENTRADAS Y LAS SALIDAS
inter s *inter
MEMORIA
int i
diag_s *diaux
i<.iinterf.n
si fun
Seleccion de la funcion deseada
.interf[i].fun
di
diag s *di
= diaux
Todos los niveles
dib22
Ordenar las entradas y las salidas
diaux
.interf[i].inter
.interf[i].inter
i=0;i++
inter
dib222
Ordenar las que vienen de niveles inferiores ya ordenadas
inter
inter
dib223
Ordenar las funciones con prototipo
inter
inter
dib224
Ordenar las entradas y las salidas que faltan
di
inter
inter s *inter

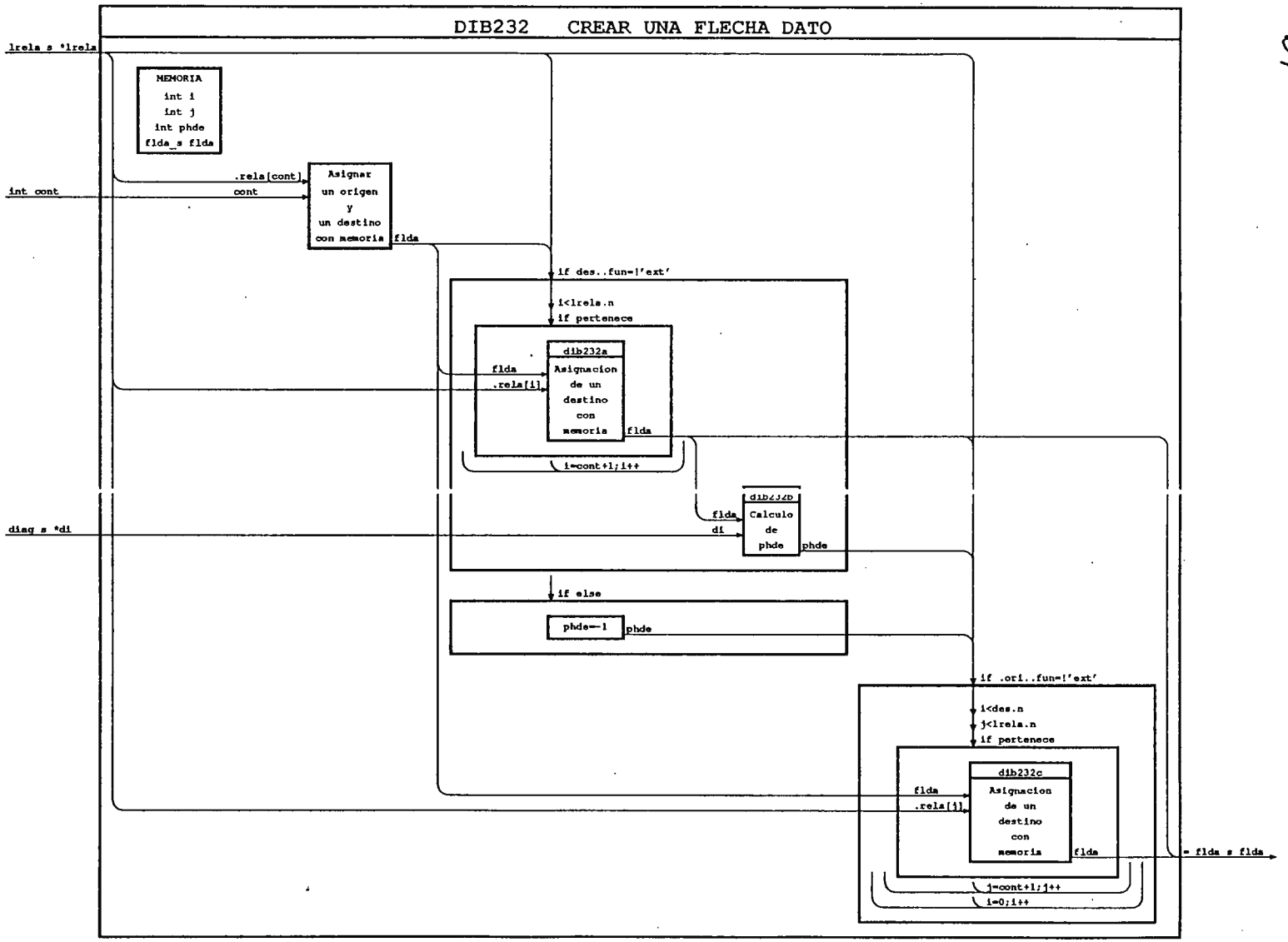
DIB224 ORDENAR LAS ENT. Y LAS SAL. QUE FALTAN

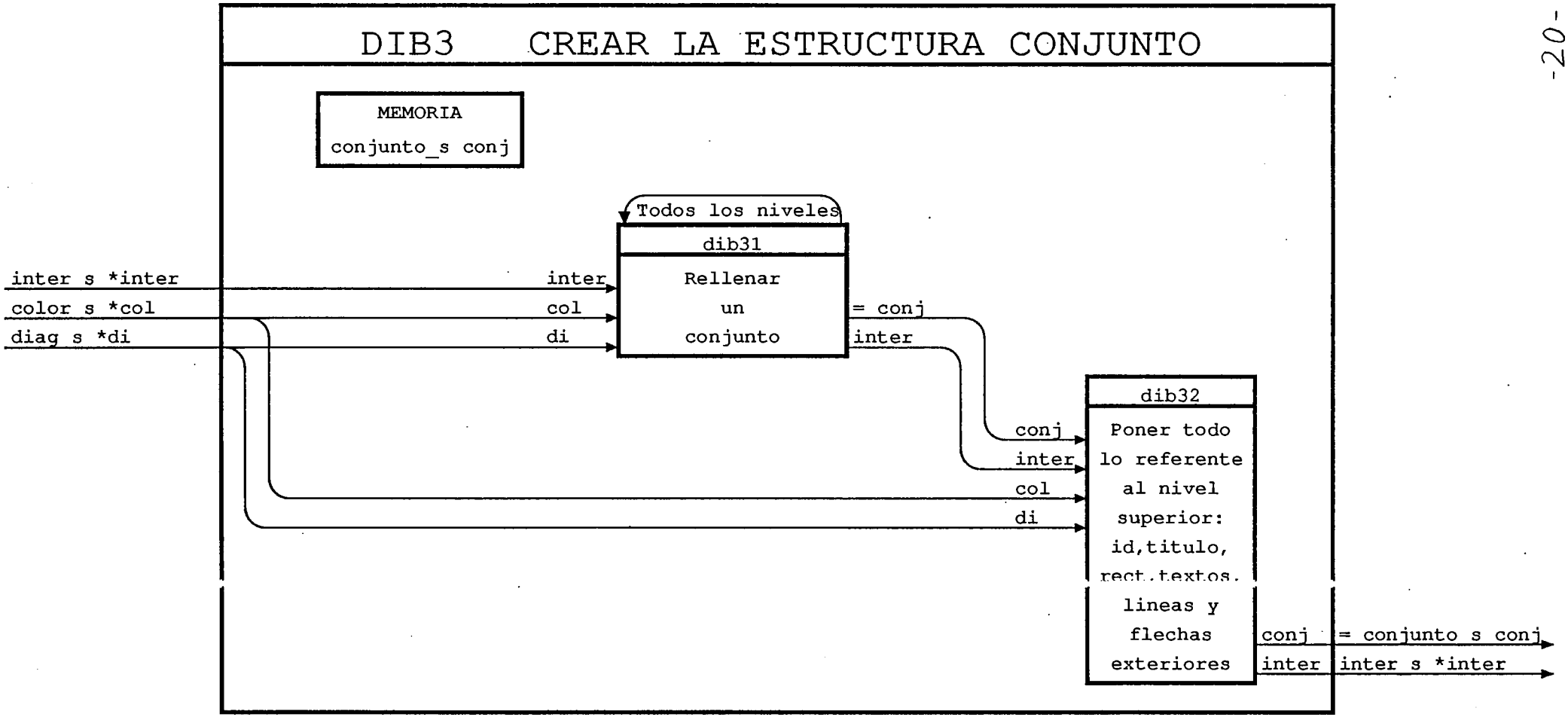


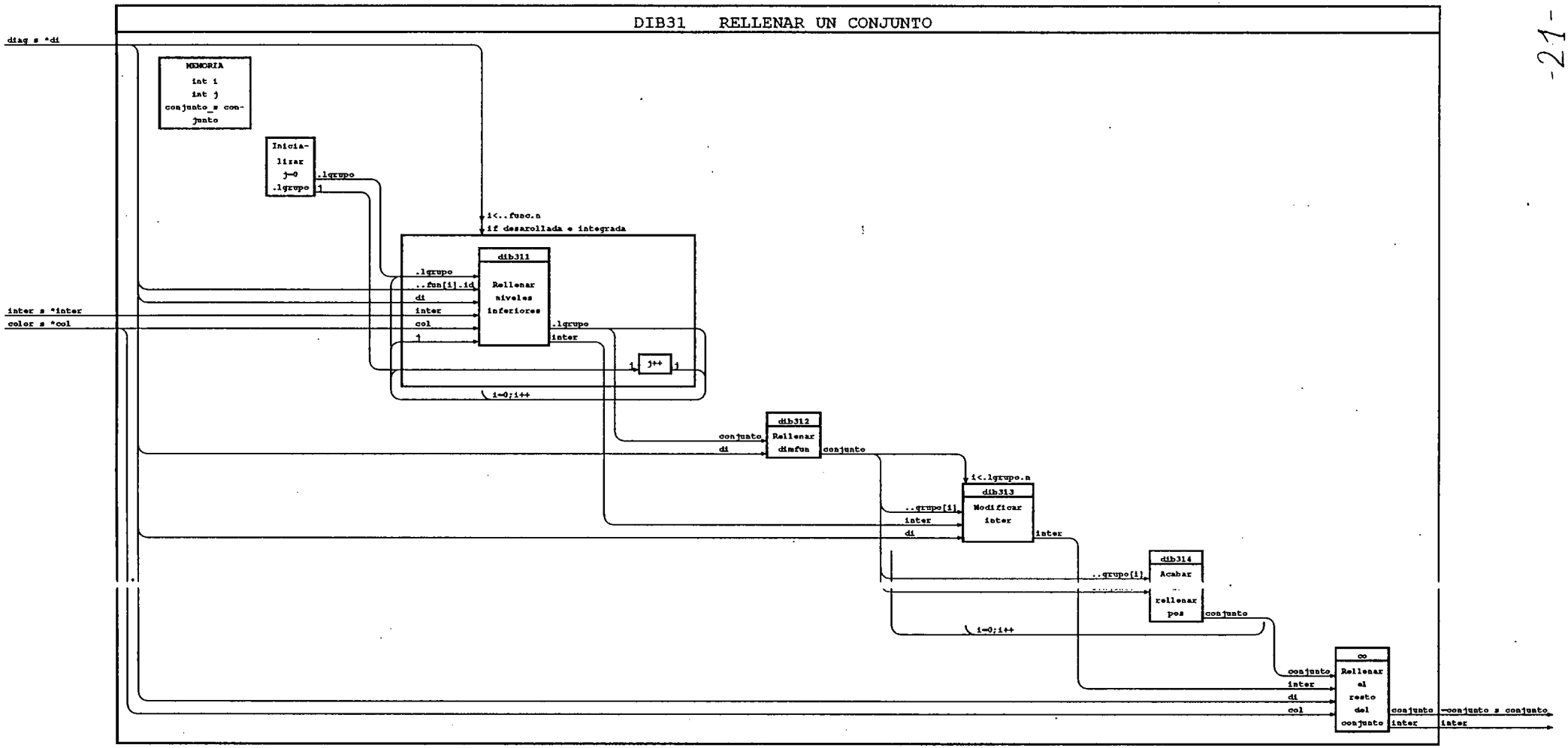

DIB224C COMPARAR DOS SALIDAS
ensa s *sal1
ensa s *sal2
char *id
macro s *macro
liof s *entradas
MEMORIA
int i
int j
int pto1
int pto2
a_d_e x
dib224c1
Casos particulares
sal1
sal2
id
.func
= x
if x=IGUAL
Return
x
dib224c2
Partido entre dos destinos
i<sal1.orde.n
j<sal1.orde.n
.od[i]
.nombre1
.od[j]
.nombre2
id
macro
entradas
i=0;i++
j=0;j++
0
if x=ANTES
pto1++
pto1
if x=DESPUES
pto2++
pto2
if pto1=|pto2
Procesar y Return
if else
Decidir por orden alfabetica
.nombre1
.nombre2
= a d e x


DIB224C2 PARTIDO ENTRE DOS DESTINOS
orides s *od1
orides s *od2
char *id
char *nom1
macro s *macro
char *nom2
liof s *entradas
dib-.a
id: dib224c21
Calcular
la bajada
del
dato
od1
id
nom1
.macro
phb1
dib-.b
id: dib224c21
Calcular
la bajada
del
dato
od2
id
.macro
nom2
phb2
if od1.tipo=od2.tipo
if od1.tipo='a'
dib-.c
id: dib224c22
Decidir
entre
dos destinos
tipo 'a'
phb2
phb1
= x
if od1.tipo='i'
dib-.d
id: dib224c23
Decidir
entre
dos destinos
tipo 'i'
phb2
phb1
od1
od2
.lfunc
entradas
= x
if od1.tipo='b'
dib-.e
id: dib224c24
Decidir
entre
dos destinos
tipo 'b'
phb2
phb1
.macro
nom1
d1
nom2
= x
if else
dib-.f
id: dib224c25
Decidir
entre
dos destinos
de tipo
diferente
phb2
phb1
od1.tipo
od2.tipo
= x
= s d e x

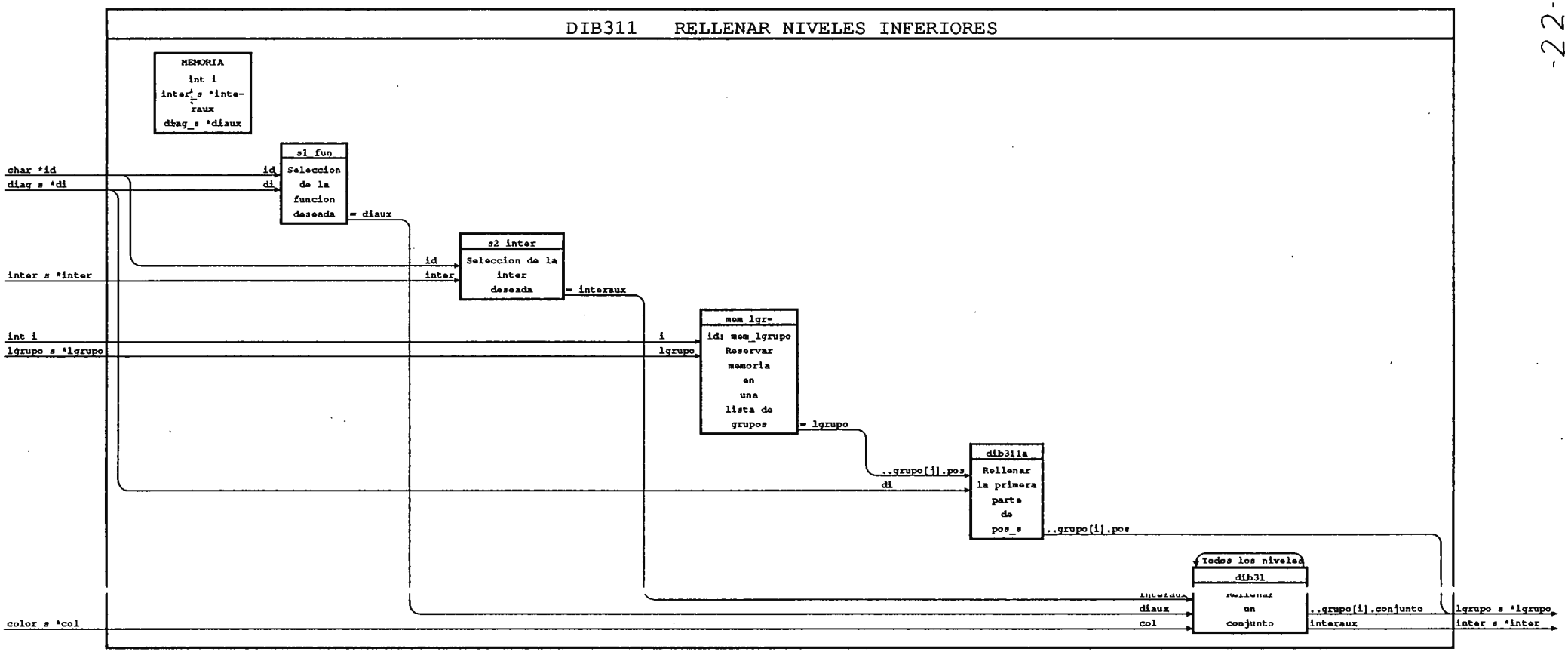

DIB23 CREAR LAS FLECHAS DATO
inter s *inter
MEMORIA
int i
lrela_s lrela
flda_s flda
diag_s *diaux
diag s *di
di
dib231
Crear las relaciones
lrela
i<lrela.n
if .rela[i].usada=NO
dib232
Crear una flecha dato
lrela
di
i
= flda
flda
.lfledat
dib233
Rellenar una flecha dato en inter y liberar flda
.lfledat
i=0;i++
fre lre-
id: fre_lrela
Liberar memoria de una lista de relaciones
lrela
i<.linterf.n
sl fun
Seleccion da la funcion deseada
.interf[i].fun
di
= diaux
Todos los niveles
dib23
Crear las flechas dato
diaux
.interf[i].inter
.interf[i].inter
inter
i=0;i++

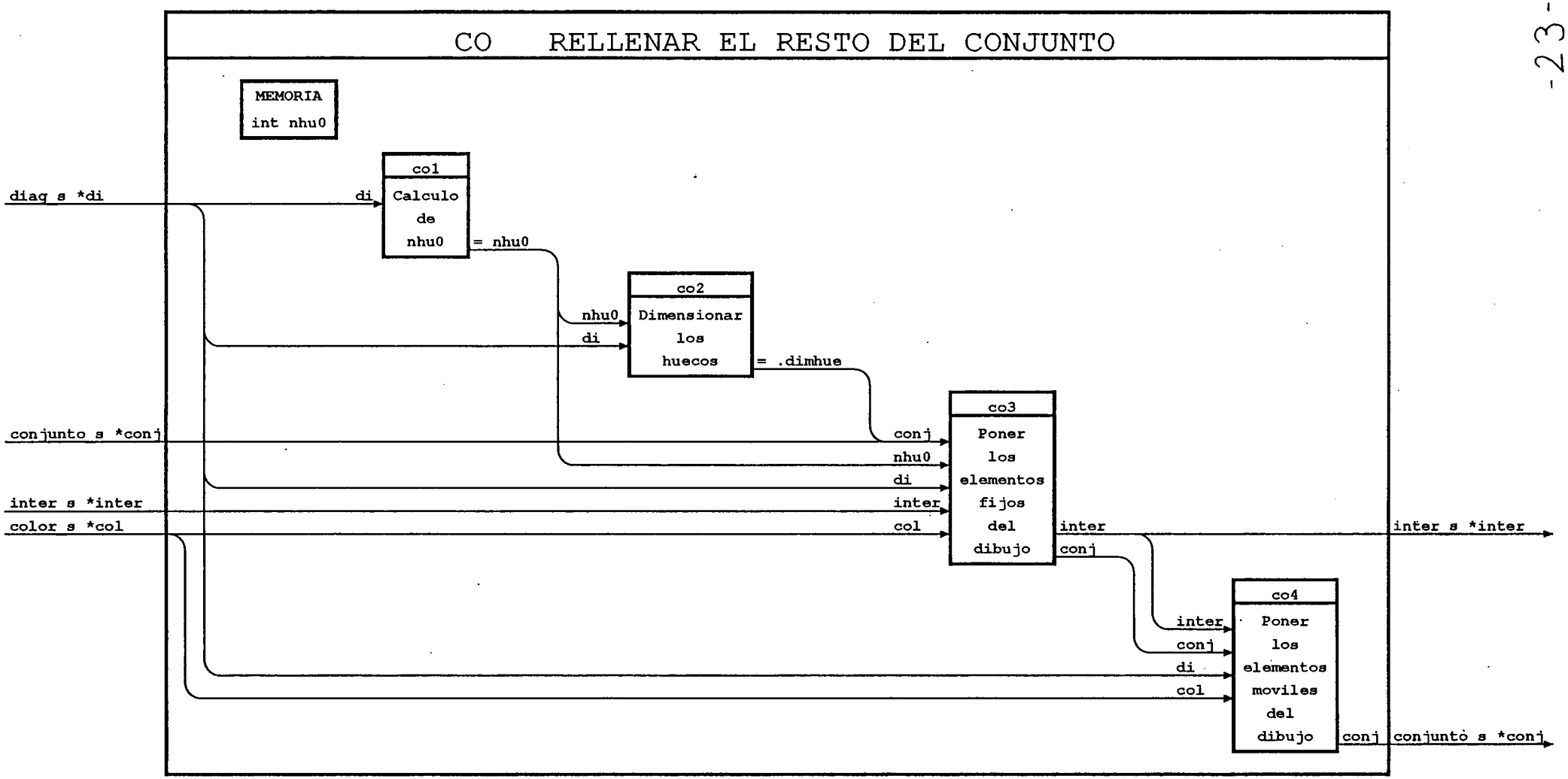

DIB232 CREAR UNA FLECHA DATO
lrela s *lrela
MEMORIA
int i
int j
int phde
flda_s flda
int cont
.rela[cont]
cont
Asignar
un origen
y
un destino
con memoria
flda
if des..fun=!'ext'
i<lrela.n
if pertenece
dib232a
flda
.rela[i]
Asignacion
de un
destino
con
memoria
flda
i=cont+1;i++
dib232b
flda
di
Calculo
de
phde
phde
diag s *di
if else
phde=-1
phde
if .ori..fun=!'ext'
i<des.n
j<lrela.n
if pertenece
dib232c
flda
.rela[j]
Asignacion
de un
destino
con
memoria
flda
= flda s flda
j=cont+1;j++
i=0;i++

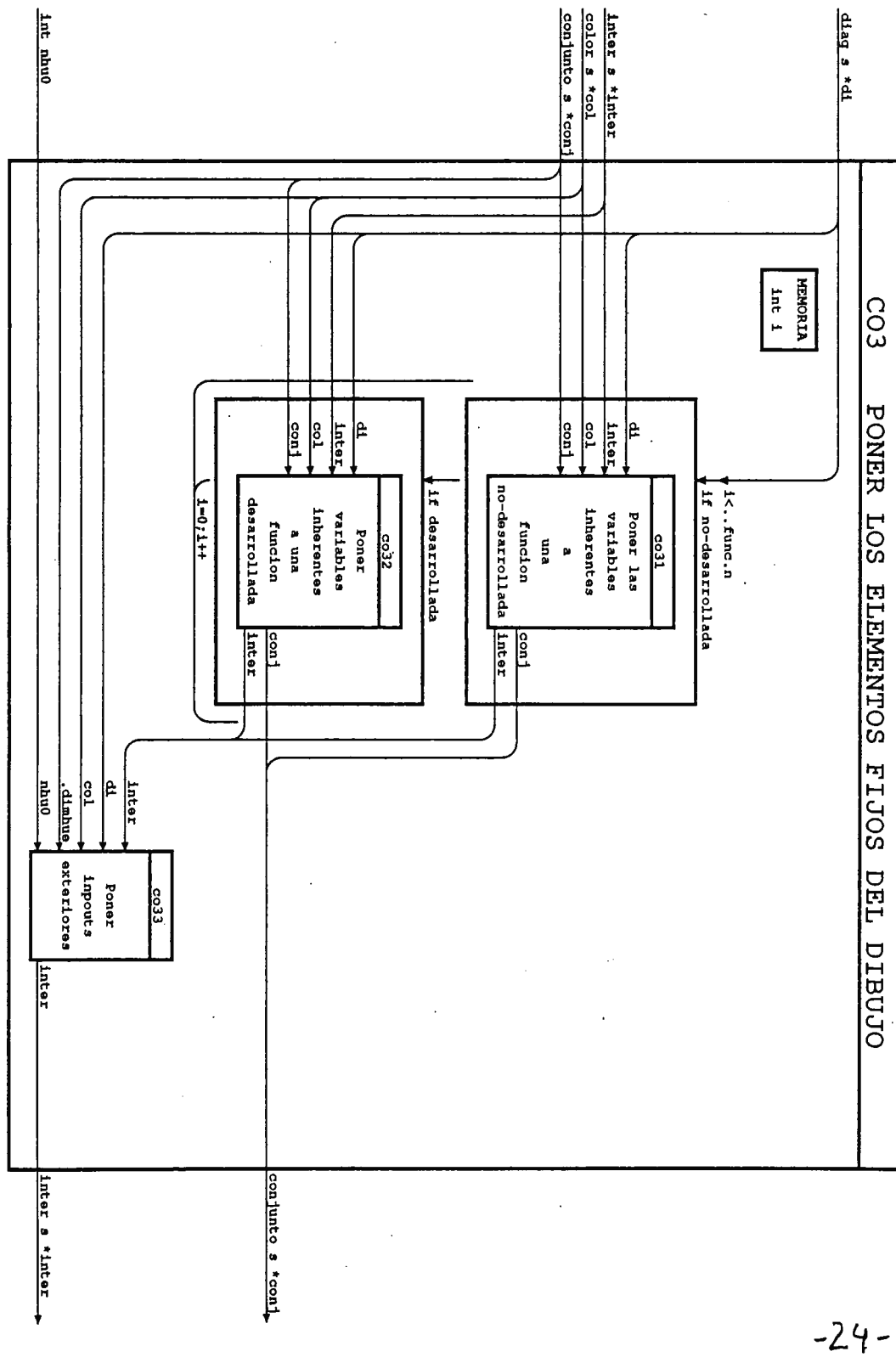

DIB3 CREAR LA ESTRUCTURA CONJUNTO
MEMORIA
conjunto_s conj
Todos los niveles
dib31
Rellenar un conjunto
inter s *inter
color s *col
diag s *di
inter
col
di
= conj
inter
dib32
Poner todo lo referente al nivel superior: id,titulo, rect.textos, lineas y flechas exteriores
conj
inter
col
di
conj = conjunto s conj
inter inter s *inter

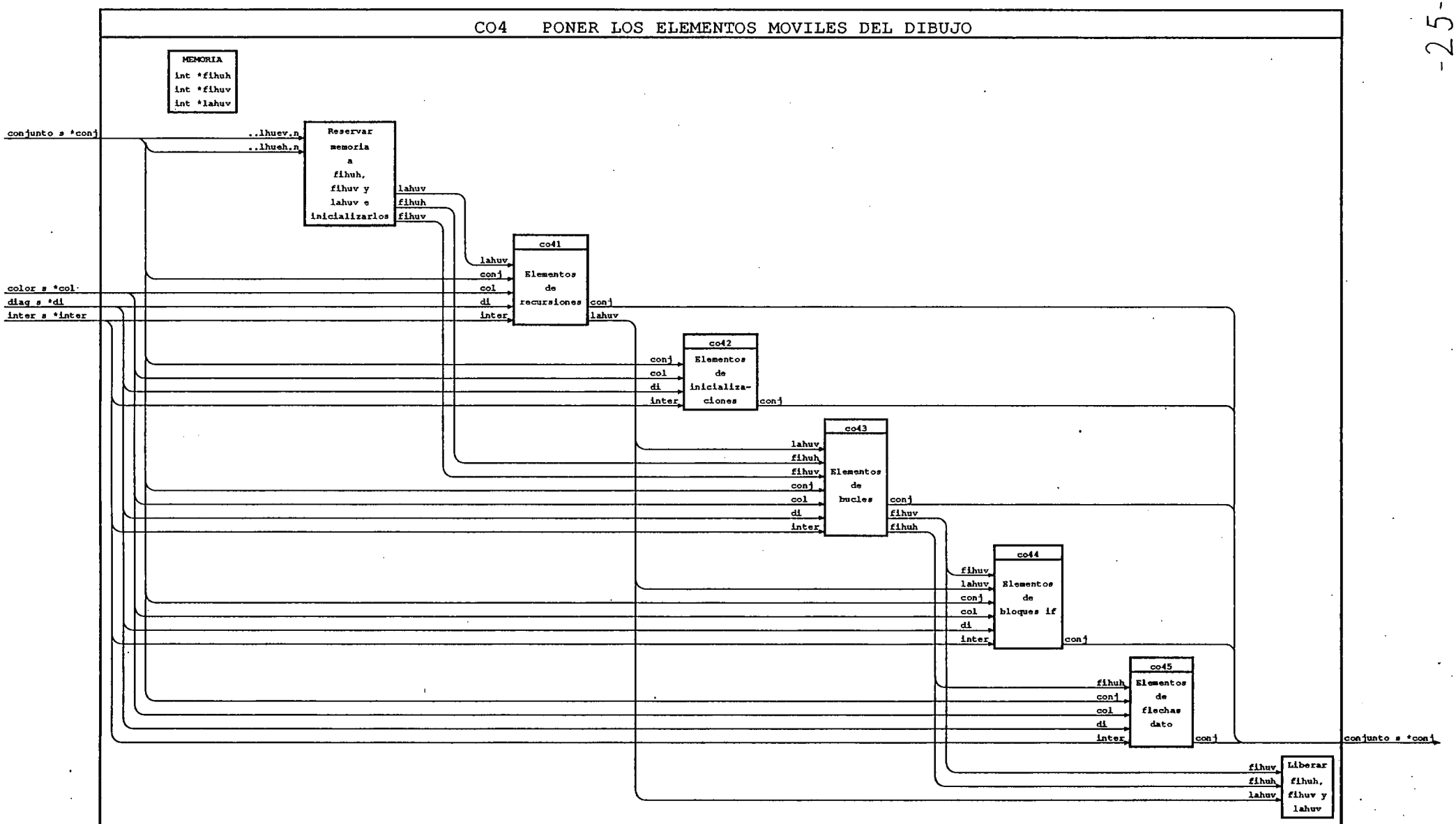

DIB31 RELLENAR UN CONJUNTO
diag s *di
MEMORIA
int i
int j
conjunto_s con-
junto
Inicia-
lizar
j=0
.lgrupo
inter s *inter
color s *col
i<..func.n
if desarollada e integrada
dib311
Rellenar
niveles
inferiores
.lgrupo
..fun[i].id
di
inter
col
1
j++
i=0;i++
dib312
Rellenar
dimfun
conjunto
i<.lgrupo.n
dib313
Modificar
inter
..grupo[i]
dib314
Acabar
rellenar
pos
∞
Rellenar
el
resto
del
conjunto
conjunto = conjunto s conjunto
inter

DIB311 RELLENAR NIVELES INFERIORES




CO RELLENAR EL RESTO DEL CONJUNTO
MEMORIA
int nhu0
co1
Calculo de nhu0
co2
Dimensionar los huecos
co3
Poner los elementos fijos del dibujo
co4
Poner los elementos moviles del dibujo
diag s *di
di
= nhu0
nhu0
di
= .dimhue
conjunto s *conj
conj
nhu0
di
inter s *inter
inter
color s *col
col
inter
conj
inter s *inter
inter
conj
di
col
conj
conjuntò s *conj


CO3 PONER LOS ELEMENTOS FIJOS DEL DIBUJO
diag s *dl
int nbu0
inter s *inter
color s *col
conjunto s *conj
MEMORIA
int i
i<..func.n
if no-desarrollada
co31
Poner las variables inherentes a una funcion no-desarrollada
dl
inter
col
conj
inter
if desarrollada
i=0;i++
co32
Poner variables inherentes a una funcion desarrollada
dl
inter
col
conj
inter
inter
dl
col
.dimbue
nbu0
co33
Poner inputs exteriores
inter
conjunto s *conj
inter s *inter


CO4 PONER LOS ELEMENTOS MOVILES DEL DIBUJO
MEMORIA
int *fihuh
int *fihuv
int *lahuv
conjunto s *conj
color s *col
diag s *di
inter s *inter
..lhuev.n
..lhueh.n
Reservar memoria a fihuh, fihuv y lahuv e inicializarlos
lahuv
fihuh
fihuv
co41
Elementos de recursiones
conj
col
di
inter
lahuv
co42
Elementos de inicializa-ciones
conj
co43
Elementos de bucles
fihuv
co44
Elementos de bloques if
co45
Elementos de flechas dato
Liberar fihuh, fihuv y lahuv
conjunto s *conj

DOCUMENTO N°3

PLIEGOS DE CONDICIONES

1- Pliego de condiciones generales y económicas

Los requisitos generales de este proyecto son desarrollar e implantar en la medida de lo posible una herramienta informática de diseño e implantación de algoritmos basada en la representación de la metodología ANA.

No se planteo ningún requisito económico debido a lo que ya se ha dicho en el estudio económico: el principal objetivo del proyecto no era conseguir un producto comercial sino el utilizarlo internamente en el I.I.T.

2- Pliego de condiciones técnicas y particulares

La herramienta desarrollada debe poder utilizarse en un ordenador PC-compatible con una tarjeta gráfica VGA o superior. La impresión se realiza mediante creación de ficheros en un lenguaje de impresión estándar (por ejemplo, PostScript).

Debe ser desarrollada y especificada siguiendo puntualmente la metodología ANA para proyectos de Ingeniería del Conocimiento.

DOCUMENTO N°4

PRESUPUESTO

1- Costes de ingeniería

Una estimación del tiempo empleado para el análisis y estudio de la documentación necesaria para el proyecto es 100 horas.

Para el estudio de la parte ya realizada anteriormente se estiman otras 100 horas.

EL tiempo empleado en el diseño e implantación de la primera versión es 900 horas. Para el diseño de la segunda versión y su implantación parcial otras 500 horas.

Por lo tanto el tiempo total estimado de dedicación al proyecto es de 1.600 horas. Si consideramos un precio de 6.000 Pts la hora de ingeniero, tenemos:

Coste de ingeniería = 1.600 x 6.000 = 9.600.000 Pts

2- Coste de material

Para la realización de este proyecto se ha utilizado un ordenador personal PC-compatible con el compilador de lenguaje C "TURBO C" de Borland. Se ha utilizado una impresora láser para la impresión de los diagramas ANA.

Su precio es:

Ordenador PC-compatible 386:	250.000 Pts
Compilador TURBO C:	30.000 Pts

Por lo tanto el material suma un total de 280.000 Pts. Si se supone que el material se amortiza en 6.000 horas de funcionamiento, resulta un precio por hora de:

$$280.000/6.000 = 46,6 \text{ Pts/Hora}$$

Estimando en un 80% del tiempo total de ingeniería como empleo del equipo, obtenemos que el coste del material utilizado es:

$$1.600 \times 0,8 \times 46,6 = 59.700 \text{ Pts}$$

3. Coste total

Sumando los costes de ingeniería y los de material, obtenemos un coste total del proyecto de:

$$\text{Coste total} = 59.700 + 9.600.000 = 9.659.700 \text{ Pts}$$

Es decir nueve millones seiscientas cincuenta y nueve mil setecientas pesetas.