UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

Un editor de pantalla orientado a intêrpretes de côdigo hilvanado

Tesis que para obtener el tîtulo de

Ingeniero en computaciôn

presenta: Octavio Orozco y Orozco.

Director: Dr. Victor Guerra Ortiz.

1985.

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

Indice

Resumen

Mucho del tiempo usado en la implantaciôn de un programa de computadora que soluciona un problema, es gastado en tareas que no son parte de dicha soluciôn p.ej. ligar y recompilar. Esto en el desarrollo de sistemas tiende a ser demasiado oneroso.

Una alternativa, es el uso de un sistema de desarrollo de programas de computadora integrado por: un lenguaje de alto nivel que permita usar en forma sencilla los recursos de la mâquina, un editor de texto que explote las caracterîsticas de dicho lenguaje con el fin de agilizar la implantaciôn de la soluciôn, un sistema operativo que permita la creaciôn y mantenimiento de programas y datos en el sistema.

En este documento se describe el editor de texto que se implantô con el fin de formar parte integral de un sistema de desarrollo con las caracterîsticas arriba mencionadas.

Resumen.

El lenguaje usado para la implantaciôn del editor fuê Forth. Forht es un intêrprete hilvanado que tiene caracterîsticas que lo hacen una buena alternativa para el desarrollo de sistemas :

El sistema es transportable.

Su implantaciôn consta de dos Kbytes en côdigo de mâquina, y el resto del intêrprete estâ construido sobre sî mismo.

El intêrprete es compacto.

Las funciones primitivas del lenguaje, tienen un poder expresivo comparable al de Pascal o C, ocupan unos seis Kbytes lo que lo hace muy atractivo para computadoras chicas.

Permite la compilaciôn incremental.

Lo cual hace innecesario compilar (y ligar) todo el programa por cada cambio que se le haga. De hecho no es una compilaciôn en el sentido de una traduc ciôn a lenguaje de mâquina; es una traducciôn a lenguaje de una mâquina virtual, que es interpretado. No obstante, la penalizaciôn en tiempo de ejecucion es muy poca con respecto al côdigo: dos a uno.

La compilaciôn es râpida.

En el peor de los casos requiere unas cinco veces menos tiempo que un lenguaje tipo Pascal.

Es conversacional.

Implica permitir al usuario la ejecuciôn de cualquier subrutina al nivel superior. Esto significa la posesiôn de un depurador dentro del mismo lenguaje. Implica tambiên la capacidad de crear nuevas subrutinas en vivo haciendo que el usuario personalice el sistema y lo adapte a su problema con facilidad.

El editor explota, en su implantaciôn, varias caracterîsticas de Forht:

Compilaciôn incremental.

Hace que esto sea un proceso bastante natural al compilar a partir de el lugar que elusuario indique, dentro del programa.

Conversacional.

La depuraciôn de cualquier rutina despuês de editada es inmediata ya que el proceso: ediciôn-compilaciôn se simplificô al mâximo.

Extensibilidad.

El editor, al estar escrito en Forth hereda êsta caracterîstica. Esto implica el permitirle al programador adaptar el editor a sus necesidades.

Adembs posee las caracterîsticas que hacen de un editor de pantalla la herramienta mâs ûtil del programador:

Râpidez

El tiempo de respuesta es inmediato para la mayorîa de las instrucciones.

Confiabilidad

Capacidad de Respaldo.

Para minimizar la posibilidad de borrado accidental o destrucciôn de un documento.

Facilidad de arrepentirse.

Permite arrepentirse, es decir, volver a su estado original algun cambio realizado. Esta facilidad se desarrollô en su mâs bâsica versiôn.

Ergonomîa

Documentaciôn en lînea.

El usuario nuevo u ocasional puede consultar en lînea un pequeño resumen de la sintâxis de las instrucciones e inclusive tener acceso completo a la versiôn del editor que estâ usando.

Retroalimentaciôn

Existe retroalimentaciôn al especificar el radio de influencia de las operaciones, asî como en los resultados de una operaciôn dada.

El editor ha servido, hasta la fecha, para dar apoyo en la creaciôn y depuraciôn de diversas aplicaciones tales como: editor de grâficas, desarrollo de cursos para enzeñanza asistida por computadora, manejadores de impresoras, controladores y diversos tipos de circuitos VLSI.

1. Introducciôn.

1.1 Sistemas de Ediciôn Interactivos

Los editores interactivos se han convertido en un componente esencial de cualquier lugar en el que se usen mâquinas computadoras. Estos programas usan el poder de la computadora para la creaciôn, ediciôn, borrado y modificaciôn de textos tales como instrucciones de programas, texto manuscrito y datos nûmericos. Un editor permite que un texto sea modificado y corregido maŝ râpido y mâs fâcilmente, en muchos ordenes de magnitud, de lo que se serîa si se tuviese que hacer la correcciôn manualmente.

A pesar de que los editores siempre han sido herramientas importantes en los sistemas de computaciôn, ha sido hasta ahora que han empezado ha convertirse en tema de investigaciôn, conforme se empiezan a convertir en componentes claves de las oficinas del futuro. Actualmente los editores no son vistos ya como herramientas de uso exclusivo de los programadores, o para secretarias que transcriben lo que les pasa en borrador de papel el autor. ahora se esta comprendiendo que el editor debe ser considerado la interfaz primaria entre la computadora y todo aquel trabajador cuyo quehacer intelectual involucre la composiciôn, organizaciôn, estudio, y manipulaciôn de informaciôn basada en computadora.

1.1.1 Panorama general

1.1.1.1 El proceso de Ediciôn

Un editor es un programa de computadora que permite al usuario crear y modificar un documento en forma interactiva.

Editar, entonces, se podrâ definir como un diâlogo interactivo entre la computadora y el usuario. Este diâlogo consiste bâsicamente en lo siguiente:

1. Selecciôn de la parte del documento que serâ vista y manipulada.
   - Viajar a travês del documento y filtrarlo para controlar lo que se podrâ ver y manipular.
2. Determinar el formato con que se presentarâ el documento en lînea y mostrarlo.
   - La presentaciôn deberâ ser la misma que se imprima en papel.
3. Especificar y ejecutar operaciones que modifiquen el documento.
   - El conjunto de operaciones que crean o modifican el documento.

4. Actualizar la presentaciôn adecuadamente.

   - Retroalimentar al usuario en tiempo real y en forma adecuada en en funciôn de los operaciones que ejecute.

1.1.1.2. El editor desde el punto de vista del usuario

Al usuario de un editor interactivo se le presenta un modelo conceptual del sistema y una interfaz de usuario.

1. El modelo conceptual del sistema es la estructura en la cual el editor y el mundo en el que opera estan basados. Sus caracterîsticas principales deberân ser :

   - Proveer una abstracciôn fâcilmente comprensible de el documento que se edita y sus elementos, asî como de un marco de referencia que permita anticipar los efectos que produce el operar sobre estos elementos.

   - Tener una estructura consistente y completa para usar e implantar el sistema.

2. La interfaz de usuario es el conjunto de herramientas y têcnicas con las que el usuario se comunica con el editor y consiste bâsicamente de :

   - Dispositivos de entrada

     Teclados, bastones (joysticks), pantallas sensibles al tacto, ratones (mice), reconocedores de voz, etc.

   - Dispositivos de salida

     Monitores: sencillos, de alta resoluciôn, etc.
     Bocinas para voz sintêtizada o tonos.

- Lenguaje interactivo

El lenguaje de interacciôn de un editor de texto puede ser divido en tres partes:

- Componente semântico

Especifica funcionalidad: Quê operaciones son vâlidas para cada elemento (caracteres,renglones,pâginas), quê informaciôn se necesita para la manipulaciôn de cada elemento, cuales son los resultados de las operaciones y cuales errores pueden ocurrir.

Define el significado de la operaciôn especificada, no estructuras de datos particulares o diâlogos para para implantar dichas operaciones. Por ejemplo: para la instrucciôn busca:

busca cadena <cr>

Busca "cadena" en el texto y si la encuentra despliega a partir de donde se encuentre. Si no, regresa a donde estaba.

- Componente Sintâctico

Especifica las reglas de entrada/salida por medio de las cuales los elementos atômicos de el lenguaje, "tokens" , se pueden agrupar para formar oraciones en la gramâtica del lenguaje.

En têrminos de entrada las oraciones formadas podrîan incluir: Cadenas de caracteres; instrucciones; posiciones dentro de la pantalla.

En têrminos de salida las oraciones formadas podrîan incluir: Cadenas de caracteres, lineas y pârrafos formateados.

La sintâxis debe ser fâcil de aprender y recordar, y debe seguir en forma natural el modelo conceptual.

- Componente Lexicogrâfico

Especifica la forma en que los lexemas, informaciôn que viene de los dispositivos de entrada o va hacia los dispositivos de salida, serân combinados para formar los "tokens" que usa el componente sintâctico.
Por ejemplo tecleando el lexema: ^B da como resultado que el carâcter que sigue al cursor sea borrado y el resto de la lînea sea recorrido a la izquierda un carâcter.

El tipo de lenguaje de interacciôn que se tiene lo define la forma en que se especifican los lexemas. Por ejemplo: Lenguaje orientado a instrucciones, a teclas con funciôn, a menû.

Es interesante hacer notar que êsta divisiôn, permite la independencia con respecto a la sintâxis, ya que diferentes estilos sintâcticos pueden ser mapeados en las mismas operaciones semânticas.

- Documentaciôn adicional

  Es deseable que contenga:

  - Descripciôn del modelo conceptual.
  - Descripciôn de la arquitectura del sistema, con terminologîa a nivel de usuario.
  - Una guîa del usuario detallando la sintâxis y semântica del lenguaje de interacciôn.
  - Una guîa de uso que posea definiciones operacionales y demuestre situaciones tîpicas con ejemplos.

1.1.1.3. El usuario desde el punto de vista del editor.

Cada individuo forma un modelo de usuario personal de un editor y este modelo puede diferir de el modelo conceptual de usuario en varias formas:

1. Como subconjunto del modelo conceptual de usuario.

   Cuando el usuario solo usa un subconjunto de instrucciones del editor.

2. Como una extensiôn de el modelo conceptual de usuario.

   Cuando el usuario hace, en forma consistente, uso de isntrucciones "primitivas" para realizar operaciones comunes en formas que no fuerôn originalmente abarcadas por el modelo conceptual.

3. Como un equivalente operacional pero lôgicamente diferente del modelo conceptual de usuario.

   Cuando el usuario tiene un modelo conceptual, del sistema, operante pero que sin embargo difiere del modelo conceptual en el que se basô el diseñador del editor.

   Es importante hacer notar aquî que el usuario hace uso del editor en base a su propio modelo conceptual del sistema.

1.1.1.4. El editor desde el punto de vista del sistema

Los editores en general siguen una arquitectura similar a la que se muestra en la siguiente figura:

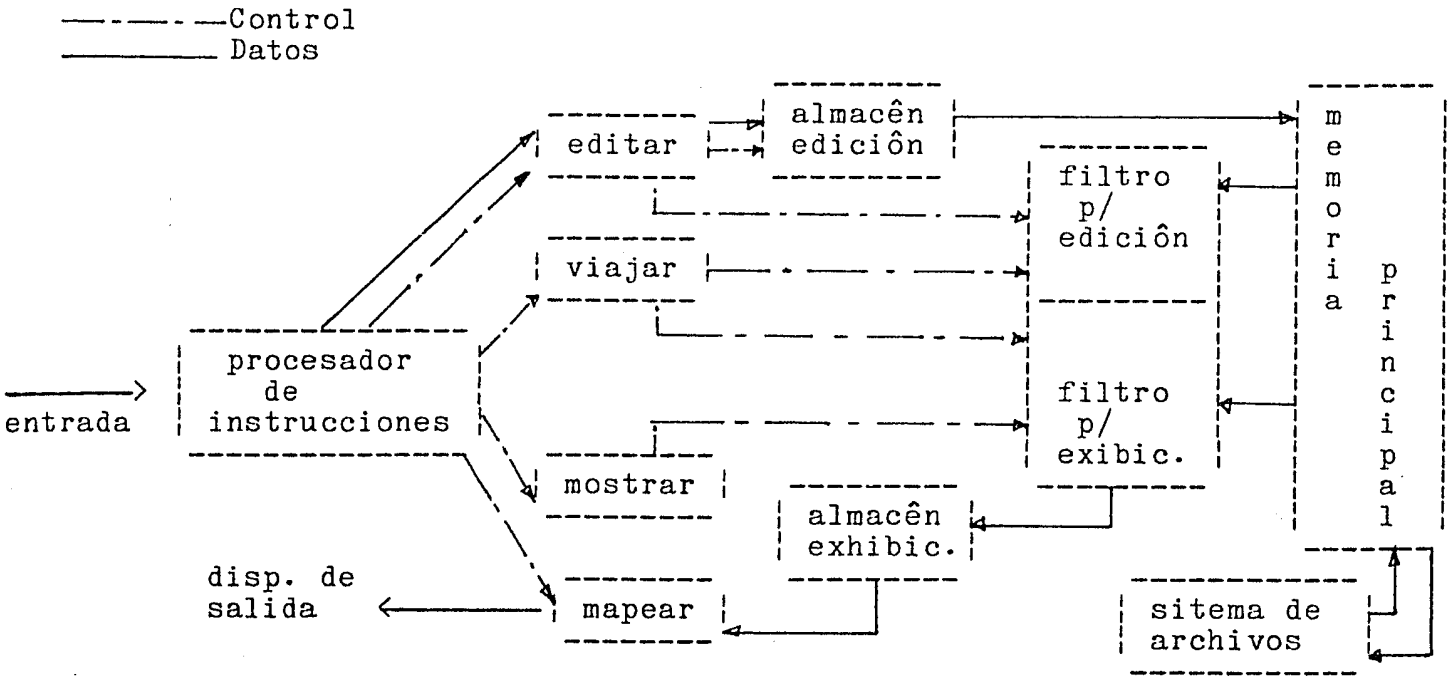


Fig. 1.1 Arquitectura general de un editor (tomado de [Mey 82])

1.1.1.4.1. Arquitectura

1. Procesador de Instrucciones:

   - Acepta datos cuya fuente son los dispositivos de entrada.
   - Analiza el lêxico y convierte la cadena de entrada en descriptores (separador, operador, identificador, etc.)
   - Analiza sintâcticamente el conjunto de descriptores y si encuentra una composiciôn vâlida de descriptores, invoca a las rutinas semânticas apropiadas.
     Las rutinas semânticas invocan a las rutinas que permiten editar, viajar, mostrar y mapear en algûn dispositivo de salida el documento.

Mientras que las operaciones de edición son siempre especîficadas explicîtamente por el usuario, y las operaciones de mapeo lo son en forma implîcita por las otras tres categorîas de operaciones, las operaciones de viajar y mostrar pueden ser explîcitamente especîficadas o implicîtamente invocadas por las operaciones de edición.

De hecho la relación entre las tres clases de operaciones puede ser considerablemente mâs complicada que el simple modelo de la sección anterior (Viajar para determinar el l u g a r que se seleccionarâ, filtrar para seleccionar q u e es lo que se va a mostrar y manipular, formatear para determinar c o m o aparecerâ el mapa del documento en los dispositivos de salida, y después editar y reformatear).

En particular, no es necesario que haya una simple relación uno a uno entre lo que se está mostrando en los dispositivos de salida y lo que se puede editar.

Para ilustrar lo anterior observaremos mâs de cerca los componentes restantes de la figura anterior.

2. Editar

Se encarga de determinar el ârea que puede ser editada, usando, lo que llamaremos un apuntador para edición. Este apuntador puede ser explicîta o implicîtamente posicionado por el usuario o por el sistema, respectivamente, como resultado de las instrucciones de edición.

3. Viajar

Posiciona los apuntadores siguientes:

El de edición.

El de exhibición.

por lo tanto êste componente marca el inicio de el punto en que el filtrado del documento empieza. (filtrado para edición y filtrado para exhibición respectivamente.)

4. Mostrar

Se encarga de determinar el ârea que puede ser mostrada, usando, lo que llamaremos un apuntador para exhibición.

Este apuntador puede ser explicîta o implicîtamente posicionado por el usuario o por el sistema, respectivamente, como resultado de las instrucciones de edición.

5. Mapear

Se toma lo que hay en el almacên de exhibiciôn y se mapea uno-a-uno a el dispositivo de salida.

6. Almacên de ediciôn

Es aquî donde se tiene una copia de la parte del documento que se esta actualizando.

7. Almacên de exhibiciôn

Aquî se tiene almacenada la copia del documento actualizada que se mostrarâ al usuario.

8. Filtro para la ediciôn

Este filtro es invocado siempre que el usuario teclea una instrucciôn. Y lo que hace es poner una parte de el documento, una copia, en el almacên de ediciôn teniendo como referencia los apuntadores de ediciôn y sus propios parâmetros. (Estos parâmetros son especificados tanto por el usuario como por el sistema. Y dan informaciôn como: Que rango puede ser afectado por el usuario, es decir, la lînea actual en un editor de lînea y la pantalla actual en un editor de pantalla).

9. Filtro para exhibiciôn del documento

Este filtro es invocado siempre que lo que se esta mostrando al usuario necesite ser actualizado.
Este filtro se encarga puês, de filtrar una copia del documento, y pasarla al almacên de exhibiciôn. (En funciôn del apuntador actual de exhibiciôn y de sus propios parâmetros).

10. Memoria principal y Sitema de archivos

Generalmente se almacena aquî una copia del documento sin embargo cuando esto es imposible o no deseable entonces se mapea el archivo completo en memoria virtual y se deja que el sistema operativo realice la paginaciôn para el editor. Las cuales se almacenarân en memoria principal hasta que una operaciôn del usuario requiera otra pâgina.

En cualquier caso los documentos son frecuentemente representados no como cadenas secuenciales de caracteres, sino como una "estructura de datos para editor", cuya caracterîstica distintiva es permitir la adiciôn, borrado y modificaciôn de caracteres minimizando el uso de las rutinas de entrada/salida asî como el movimento de caracteres.

1.1.1.4.2 Configuraciones

Los editores funcionan en tres tipos bâsicos de ambientes de computaciôn a saber: Tiempo compartido, dedicado (personal), y distribuido.

1. Tiempo compartido

   Este tipo de editor debe funcionar adecuadamente en el contexto de la carga impuesta al procesador de la computadora, a la memoria principal y a la memoria secundaria.

2. Dedicado

   En este caso el editor debe tener acceso a las funciones que el editor de un sistema de tiempo compartido obtiene del sistema operativo de la computadora anfitriôn.

3. Distribuido

   El editor en este tipo de sistemas debe, como en el caso de un editor para un sistema personal, correr en forma independiente en cada una de las mâquinas que conforman la red, y ademâs debe competir, como en el caso de un editor de un sistema de tiempo compartido, por recursos tales como: archivos.

1.1.2. Desarrollo de los editores

Esta parte consiste, bâsicamente, de un panorama general en cual se revisan algunos conceptos importantes.

1.1.2.1 Ediciôn computarizada no interactiva

La unidad bâsica fuê la lînea de 80 columnas; el usuario hacîa correcciones lînea por lînea, reescribiendo las tarjetas errôneas.

1.1.2.2 Ediciôn de tarjetas

Aquî el conjunto inicial de tarjetas de los usuarios se almacenaba en un archivo imagen, ya sea en cinta o disco magnêtico. Cada tarjeta se referenciaba por un nûmero de secuencia ûnico. Los cambios se hacîan creando un conjunto de tarjetas de ediciôn compuesto por las tarjetas que contenîan las intrucciones de ediciôn y se corrîa el conjunto de tarjetas usando un programa editor que corrîa fuera de lînea. Los editores de este tipo, que corrîan fuera de lînea, eliminaron los problemas de tarjetas que resultaban de desecho asi como el de tener que reescribir, y en algunas versiones permitieron operaciones como las de hacer reemplazos globales de un patron dado.

Sin embargo, habîa ciertos inconvenientes, por ejemplo: los programadores debîan tener un listado del conjunto de tarjetas completo antes de tratar de hacer algûn cambio. Y algunas de las ventajas organizacionales de las tarjetas se perdierôn tales como la inspecciôn visual, caracterizada por su sencillez, de las siguientes caracterîsticas:

- Secuencia adecuada
- Côdigo de colores
- Etiquetado adecuado de cajas de tarjetas.

1.1.2.3. Editores de lînea interactivos

Se diseñaron a mediados de los 60 ^s, con el advenimiento de los sistemas de tiempo compartido.

Las caracterîsticas que vale la pena remarcar de estos editores son las siguientes:

- Permitîan crear y modificar archivos desde una terminal.
- Las lîneas de estos archivos eran de longitud fija, inicialmente de 80 caracteres.
- Muchos de estos editores permitîan hacer correcciones, usando mucho de la sintâxis que caracteriza a sus antecesores. Era tîpico que muchos de estos editores compartierân la infortunada propiedad de t r u n c a r: si una inserciôn o un cambio forzaba a la lînea a exceder su longitud mâxima, ! los caracteres se tiraban por el fin de la lînea segûn se fuese necesitando !. Esta "caracterîstica" de implantaciôn, parte de un modelo conceptual de el proceso de ediciôn basado en la simulaciôn de tarjetas perforadas y listados de impresoras de lînea, lo cual fuê solo marginalmente aceptable para la ediciôn de programas y completamente inaceptable para la creaciôn de manuscritos en serio. (En este ûltimo caso, la creaciôn automâtica de una lînea en el caso de rebosamiento de caractes hubiese sido un problema cuya soluciôn es realmente trivial).

1.1.2.4. Editores de lînea basados en contexto

Otro avance que se tuvo en los editores fuê la posibilidad de identificar una lînea que contenîa el "objetivo" de una operaciôn dada, sin tener que especificar el nûmero de lînea. Es decir, el poder identificar una lînea dado un patrôn, un carâcter, que el editor tenîa que encontrar.

En esta parte de la historia de los editores el usuario erâ aûn forzado a pensar en têrminos de entidades de lîneas mûltiples, tales como parrâfos y bloques de programas; no habîa instrucciones tipo interlînea que puedieran, por ejemplo, borrar texto que fuese de la mitad de una linea a la mitad de otra.

1.1.2.5. Editores de lînea de longitud variable

Este fuê el primer "rompimiento" con los editores de "tarjetas de 80 columnas". Aûn asi el elemento primario de ediciôn fuê la lînea pero ahora êsta podîa ser de longitud "arbitraria" (lîmitada a un mâximo de digamos 500 caracteres).

Es importante remarcar que el hecho de haber eliminado la restricciôn de tener una imagen de una tarjeta al estar editando, diô como resultado un impacto fuerte y benêfico en la versâtilidad de el procesamiento de texto.

Otro importante hecho, mucho tiempo despuês comprendido, es que el texto que se estâ mostrando NO tiene que ser un mapa uno-a-uno de la representaciôn interna, sino que puede ser una visiôn mâs abstracta y adecuada de los elementos editables.

Sin embargo aûn se tenîan problemas en lo que a ediciôn de manuscritos se refiere, bâsicamente:

- Truncado de la lînea cuando esta pasaba del lîmite impuesto .
- Inahabilidad para editar una cadena de caracteres que crusarâ el lîmite entre dos lîneas.
- Inhabilidad para buscar una cadena una cadena de caracteres que cruce el lîmite entre dos lîneas.

1.1.2.6. Editores de lînea infinita

Estos editores resolvieron los problemas bâsicos de los editores de lînea de longitud variable, al eliminar por completo los lîmites entre lîneas. El texto entero fuê considerado como una lînea infinita la cual es convertida en lîneas visibles por las rutinas que muestran la informaciôn.

1.1.2.7. Editores de pantalla

Se llaman tambien editores de cursor. Y permiten otra forma de eliminar las limitaciones de los editores de lînea/superlînea usando el poder de las pantallas para mostrar varias lîneas a la vez. (lo cual permite direccionar el cursor y la posibilidad de tener almacenes locales, de uso transitorio, para ediciôn).

Estos editores pueden trabajar con lîneas de longitud variable o "infinita", ofreciendo al usuario una pantalla llena de texto que puede ser editada sin tener que pensar en têrminos de una lînea de longitud fija.

1.1.2.8. Desarrollos relevantes en el campo

1. Editores orientados al autor

Es un desarrollo importante debido a que el editor se ve como una h e r r a m i e n t a p a r a e l a u t o r es decir, un medio interactivo de organizar y revisar informa - ciôn y no como una herramienta mundana para alterar caracteres en un solo archivo. (La idea fuê desarrollada a principios de 1959 en el instituto de investigaciones de Stanford en Estados Unidos de Norteamêrica por Douglas Engelbart y la implantô en un sistema llamado NLS (oN Line System)).

2. Editor dirigido por estructura/sintâxis

Con este tipo de editores los usuarios pueden concentrarse en la tarea de escribir propiamente dicha, sin preocuparse por la estructura, ya que el editor indenta y enumera , por ejemplo, en forma automâtica. Con esto se evita que el usuario tenga que aprender una sintâxis muy detallada ya que las construcciones siempre son coceptualizadas como unidades abstractas y êstas a su vez, se pueden identificar con una sola tecla.

3. Desarrollo de utîlerias para editores

Este es un desarrollo importante que se originô a principios de los 70ŝ y bâsicamente consiste en darle tanto peso a las utîlerias para ediciôn como a las utîlerias para programaciôn

Ejemplos de este tipo de utîlerias son: formateadores de texto, de ecuaciones, de tablas, de bases de datos bibliogrâficas, asî como correctores de ortogrâfia y de estilo.

4. Editores/Formateadores

El archivo del usuario se despliega en una pantalla usando , en el mejor de los casos, mapeadores de bits en un facsîmil con la tipogrâfia y diseño del documento final.

1.1.3. Conclusiones

Mâs que pretender dar un conjunto de conclusiones terminantes, se pretende sugerir un conjunto de criterios para el diseño de un editor ideal.

- Modelo conceptual

  Debe estar bien definido y ser consistente. El usuario debe estar familiarizado y agusto con la filosofîa que hay detrâs del sistema.

- Documentaciôn

  Es importante que haya documentaciôn en lînea, una instrucciôn del editor que sea a y u d a por ejemplo; y documentaciôn fuera de lînea como manuales. Este tipo de documentaciôn debe explicar el modelo conceptual asi como los detalles de la interfaz con el usuario y las funciones del sistema.

- Interfaz con el usuario

  Debe ser clara y concisa, fâcil de aprender y usar, y ademâs debe ser consistente a travês de diferentes tipos de documentos tales como: texto, grâficas y voz. De hecho, una buena forma para probar si una interfaz es eficiente y agradable es que los autores usen el sistema para componer y revisar los manuscritos por ellos mismos. (No deberân necesitar de expertos en el uso del sistema para que les guîen, o secretarias para que hagan cambios, en ninguna de las fases de creaciôn o ediciôn del documento).

- Hacer/Arrepentirse

  Una capacidad infinita para hacer y arrepentirse le permite al autor el experimentar sin tener que preocuparse por la pêrdida o daño a un documento.

- Fâcilidades

  Fâcilidades poderosas, es decir con pocas restricciones y excepciones, que permitan al usuario hacer todo lo que se puede hacer con textos de papel: usar lapîz rojo, calculadora, tijeras y cinta adesiva. Ademâs debe tomar ventaja de las capacidades de la computadora para compensar las lîmitaciones humanas. Por ejemplo: Sustituir un patrôn dado por otro en forma global a en un documento; replicaciôn de una frase de uso corriente, de un parrâfo, y renumeraciôn atomatica de secciones o referencias despuês o mientras el archivo es editado.

- Acceso a informaciôn compartida

  Que el usuario pueda accesar informaciôn y archivos compartidos bajo situaciones controladas.

- Mezclar documentos

  Debe tenerse la facilidad para mezclar diferentes documentos tales como: texto, grâficas, programas y formas con fâcilidad.

- Mûltiple contexto

  Mûltiple contexto en la misma superficie de exhibiciôn, permitiendo al usuario el revisar y usar una gran cantidad de utîlerias familiares y documentos en una sesiôn de ediciôn. El editor no deberâ forzar al usuario a un medio ambiente pequeño y menos poderoso sino que debe formar parte de un medio ambiente mayor e integrado, permitiendo al usuario, en la mitad de una sesiôn de ediciôn el obtener informaciôn mirando a travês del sistema de archivos, el usar una utîleria que emule una calculadora u obtener un mensaje de correo elêctronico o una pieza de datos de un sistema de base de datos con regreso transparente a la sesiôn de ediciôn.

- Mostrar el documento en su versiôn final

  La habilidad de editar un facsîmil muy parecido a la composiciôn, a la disposiciôn del texto, y a la tipografîa final del documento sin un impacto significante en el tiempo de respuesta de la computadora.

  Para profundizar mâs en este tema [Mey 82] es la referencia que se recomienda, ya que trata el tema de editores y formateadores en mayor detalle.

1.2 Objetivos del programa

El programa debe ser capaz de permitir la creaciôn, modificaciôn y mantenimiento de programas y textos sin ningûn tipo de formateo. No obstante explotarâ el concepto de extensibilidad, para realizar funciones tales como: formateo de textos y ediciôn de documentos que involucren grâficas y textos.

Es importante hacer notar aquî que para extender el editor no se crearâ un "lenguaje de juguete", sino que êstas se deberân poder hacer en el mismo lenguaje en que el editor sea escrito, dandole asî al usuario una herramienta realmente poderosa para poder personalizar el editor a cualquier nivel que lo desee.

La interfaz con el usuario deberâ ser clara y concisa. Las intrucciones constarân tanto de teclas de control para las funciones mâs usadas y de cadenas de caracteres, comandos, para aquellas funciones que se usen con menos frecuencia.

Deberâ tener la capacidad de "arrepentirse" de un cambio hecho en el momento a fin de permitirle al usuario el exprerimentar sin tener que preocuparse de la pêrdida o daño a un documento.

Entre las fâcilidades que deberâ poseer estân las siguientes: realizar la bûsqueda de un patrôn dado; la bûsqueda y reemplazo, en forma global, de un patrôn dado por otro y replicaciôn de parrâfos o palabras.

Se podrâ accesar informaciôn y archivos compartidos bajo situaciones controladas.

Deberâ tener la capacidad de mezclar diferentes documentos tales como, texto y grâficas con fâcilidad.

El editor no deberâ forzar al usuario a un medio ambiente pequeño y menos poderoso que el sistema en el que se usa, sino que este debe formar parte de un medio ambiente mayor e integrado, permitiendo al usuario, en la mitad de una sesiôn de ediciôn, el obetner informaciôn mirando a travês del sistema de archivos y el usar una utîleria que emule una calculadora con regreso transparente a la sesiôn de ediciôn.

2.1 Editor de pantalla

Este capîtulo describe el diseño del sistema en base a los objetivos establecidos. El capîtulo 4 explica cômo se implantô el sistema, què objetivos se satisfacieron y cuales no.

2.1.1 Modelo del editor

La parte medular en el diseño del editor consiste en lograr que êste facilite la interacciôn entre los elementos que se encargan de mostrar al usuario el documento que esta editando y aquêllos que se encargan de actualizar la representaciôn interna de dicho documento.

Para lograr esto el diseño del editor contempla siete partes principales:

- Manejo de dispositivo de entrada.
- Almacên de Ediciôn.
- Representaciôn interna del documento.
- Manejo del Archivos.
- Manejo de Almacen de exhibiciôn.
- Manejo de disopositivo de salida.
- Procesador de instrucciones.

La representaciôn grâfica del modelo del editor se muestra en la figura siguiente:

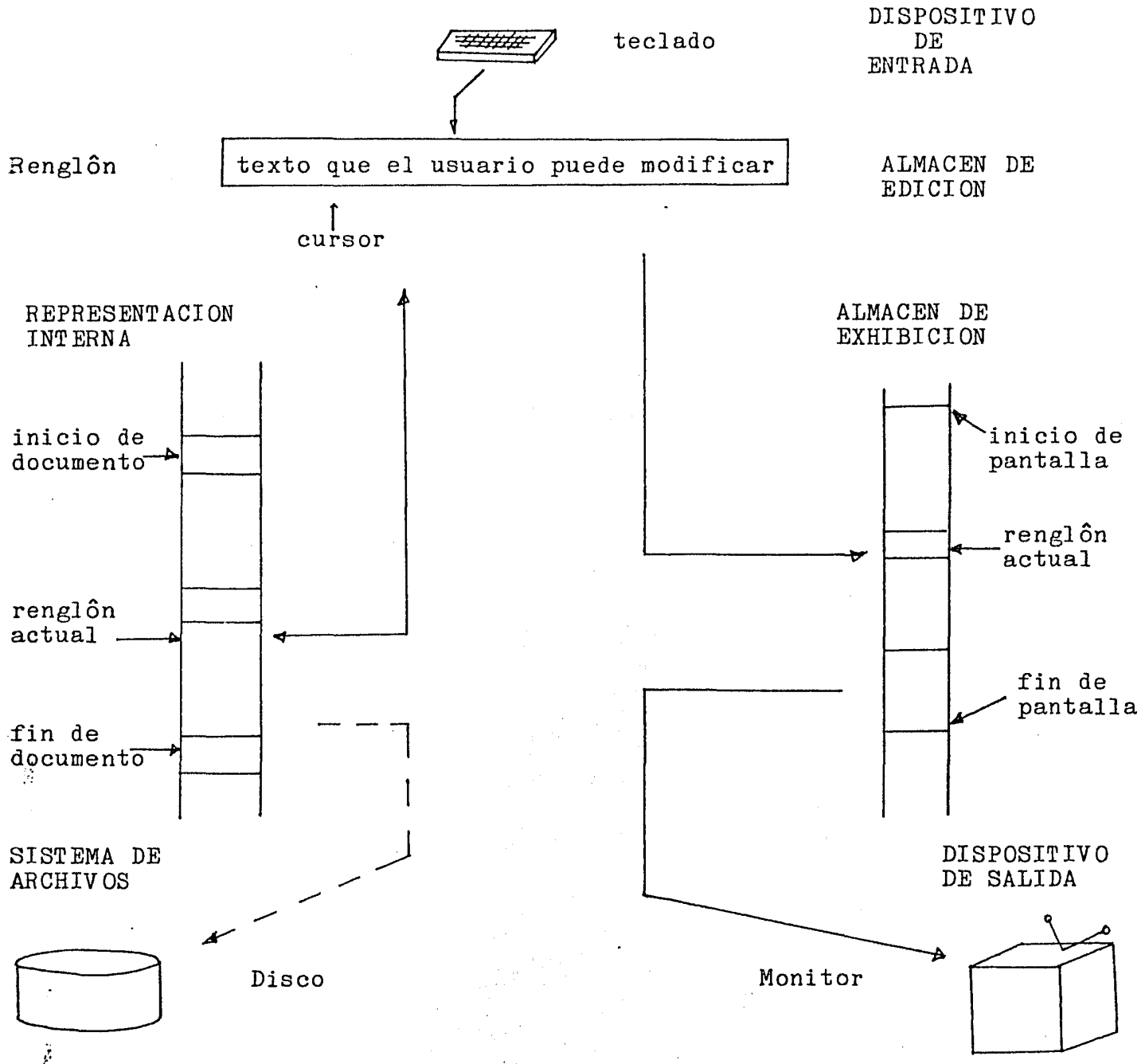


Fig. 2.1 Representaciôn grâfica del modelo del editor

2.1.1.1 Manejo de dispositivo de entrada

El manejo del dispositivo de entrada deberâ:

- Permitir al usuario definir el modo en que se deberâ procesar lo que teclea: instrucciones para el editor o entrada de texto.
- Dar la libertad al usuario de abandonar el proceso de ediciôn.

2.1.1.2 Almacên de ediciôn.

El almacên de ediciôn debe consitir de una estructura de datos con las caracterîsticas siguientes:

- Longitud variable.
- Inserciôn y borrado de algûn elemento en cualquier parte de êsta.
- Libertad de movimiento longitudinal dentro de la estructura.
- Detectar cuando se modifique el contenido de la estructura.
- Los elementos que maneje serân caracteres alfa-nûmericos.

2.1.1.3 Representaciôn interna del documento.

La representaciôn interna deberâ consistir de una estructura de datos que con las caracterîsticas siguientes:

- Longitud variable.
- Inserciôn y borrado de algûn elemento en cualquier parte de la estructura.
- Libertad de movimiento longitudinal en cualquier parte de la estructura.
- Permite la transferencia de, y hacia el almacên de ediciôn de un elemento.
- Los elementos que maneje serân renglones, copias formateadas de el almacên de ediciôn,de longitud variable.

2.1.1.4 Mantenimiento del almacên de Exibiciôn.

El almacên de exhibiciôn se actualizarâ cada vez que haya una modificaciôn en el documento. Ya que de su correcta actualizaciôn depende toda la interacciôn con el usuario.

Se pueden dar dos tipos de dispositivo de salida, con lo cual cambia la forma de tratar este almacên:

Dispositivos de salida inteligente:

En este caso no hay que mantener en memoria principal dicho almacên y basta con enviarle las instrucciones que se quieren realizar y/o datos que se desea mostrar en el dispositivo de salida.

Dispositivos de salida de refrescamiento: En este caso hay que mante-

En este caso hay que mantener en memoria principal êste almacên y hacer un manejador especial para usarlo.

En general este almacên tendrâ las caracterîsticas siguientes:

- Es una estructura de datos que contendrâ una copia de tantos elementos de la representaciôn interna como pueda contener. Su capacidad, generalmente, es de una pâgina de texto.
- Deberâ permitir el movimiento total o parcial de sus elementos hacia las partes altas de memoria o hacia las bajas. (Con esto se da la impresiôn al usuario de que el texto se puede recorrer y manipular sin muchas restricciones).

2.1.1.5 Manejo de dispositivo de salida.

- El manejo del dispositivo de salida tiene como funciôn principal:
  - Actualizar el contenido de el almacên de exhibiciôn en funciôn del lo que el usuario tecle: instrucciones o texto.
  - Detectar cuando el cursor estâ en los lîmites de la pantalla, con el fin de actualizar el contenido del almacên de exhibiciôn.

2.1.1.6 Procesador de instrucciones.

- El procesador de instrucciones debe contemplar lo siguiente:

  - Ser muy fâcil de usar.

    Esto se consigue reduciendo la cantidad de instrucciones a un nûmero razonable, tal que el usuario las recuerde fâcilmente, y haciendo que êstas tengan asociada una tecla de control como nombre.

    Las intrucciones que tendrâ el editor permitirân:

    - Movimiento del cursor

      Vertical y horizontal, en ambos sentidos.

      Posicionar el cursor al inicio o al final del renglôn.

      Posicionar el cursor en la pâgina siguiente o anterior.

    - Borrar texto

      Caracteres y lîneas.

      Con la alternativa de arrepentirse y volver al estado original.

    - Inserciôn de uno o mâs caracteres en medio de dos caracteres consecutivos.

    - Bûsqueda de una cadena con/sin reemplazo.

  - Permitir que el usario le agregue nuevas funciones sin dificultad.

2.1.1.7 Manejo de Archivos.

- El manejo de archivos consistirâ bâsicamente de una interfaz entre el editor de textos y el manejador de archivos de la mâquina. Esta interfaz debe permitir:

  - La creaciôn de archivos.

  - Acceso a un archivo, generalmente indicado por el usuario.

3. El lenguaje de programaciôn.

El intêrprete de côdigo hilvanado que se usô para elaborar el editor fuê el lenguaje Forth. Las ventajas que se obtinen de esta elecciôn son las siguientes:

- Transportabilidad

  El sistema es bastante transportable. El lenguaje empleado tiene alrededor de dos Kbytes en côdigo de mâquina, y el resto del intêrprete esta construido sobre sî mismo.

- Minimizar uso de RAM

  El intêrprete es compacto. Las funciones primitivas del lenguaje, que tinenen un poder expresivo comparable al de Pascal o C, ocupan seis Kbytes, lo que lo hace muy atractivo para computadoras chicas.

- Permitir compilaciôn incremental

  La compilaciôn no es, de hecho, en el sentido de una traducciôn a lenguaje de mâquina; mâs bien es una traducciôn al lenguaje de una mâquina virtual, que es interpretado. No obstante la penalizaciôn en tiempo de ejecuciôn es muy poca con respecto al côdigo compilado: dos a uno. De cualquier manera es innecesario compilar (y ligar) todo el programa por cada cambio que se le haga.

- Agilizar la depuraciôn

  La compilaciôn es râpida. En el peor de los casos (para Forth) requiere unas 5 veces menos tiempo que un lenguaje tipo Pascal.

  Es conversacional. Implica permitir al usuario la ejecuciôn de cualquier subrutina al nivel superior. Esto significa la posesiôn de un depurador d e n t r o del mismo lenguaje. Implica la capacidad de crear nuevas subrutinas e n v i v o haciendo que el usuario personalice el sistema y lo adapte a su problema con facilidad.

No todas estas caracterîsticas son originadas per se; algunas son resultado de la combinaciôn de Forth como lenguaje y el côdigo hilvanado como su implantaciôn. Es importante establecer la frontera entre un lenguaje y su implantaciôn. De hecho, el côdigo hilvanado empezô a usarse en el côdigo generado por un compilador de Fortran.

A continuaciôn daremos una explicaciôn de lo que es el côdigo hilvanado en abstracto y posteriormente una descripciôn de Forth. Aunque tradicionalmente cuando se especifica un sistema primero se exponen sus caracterîsticas lôgicas y luego su implantaciôn, en este caso se justifica hacerlo en el orden inverso pues Forth estâ directamente inspirado en el côdigo hilvanado.

3.1 Côdigo hilvanado

El côdigo hilvanado, bajo circunstancias adecuadas, logra un buen balance entre velocidad y poco espacio en memoria. Esta caracterîstica surge con el fin de resolver uno de los compromisos fundamentales de la ingenierîa de software: espacio en memoria contra tiempo de ejecuciôn.

3.1.1 Côdigo hilvanado directo

Las têcnicas mâs comunes para programaciôn pueden ser denotadas como: "programaciôn en lenguaje de mâquina" y "programaciôn con un lenguaje interpretado". El côdigo de mâquina es el mêtodo mâs usado de programaciôn. Cada instrucciôn del programa es escogido del conjunto alambrado en la mâquina huêsped por sus diseñadores. Cada una de esas instrucciones se ejecuta râpidamente ya que esta alambrada en la circuiterîa fîsica de la mâquina. Desde otro punto de vista, el conjunto de instrucciones dado es sub-ôptimo para casi cualquier problema especîfco, debido a que el usuario es forzado a aceptar generalidad innecesaria en ciertos lugares y "rodear" por la falta de ella en otros. En resumen, cuando se escribe en lenguaje de mâquina el usuario necesita relativamente muchas instrucciones, cada una de las cuales se ejecuta râpidamente.

Un intêrprete,en contraste,le permite al usuario el escoger su propio conjunto de instrucciones tal que corresponda a su problema especîfico. Obviamente tal libertad permite un programa mâs corto que resuelve el problema. La penalizaciôn es que el conjunto de instrucciones no estâ implantado en el hardware de la computadora. En su lugar el intêrprete debe ser un programa de computadora que simula la acciôn de el conjunto de instrucciones del intêrprete en têrminos del conjunto de instrucciones de la computadora. Esta puede ser una opciôn que consume tiempo. Por lo tanto el côdigo del intêrprete tiende a ser corto pero mâs lento que el côdigo de mâquina.

Es instructivo el mirar la relaciôn entre el hardware de la computadora huespêd y las alternativas discutidas. En el caso de el côdigo de mâquina una instrucciôn dirige el flujo de procesamiento al ejecutarse desde el IR (registro de instrucciôn) de la mâquina. En el caso de un intêrprete, una instrucciôn es de hecho un mero dato para el programa que emula al intêrprete. Por lo tanto el intêrprete dirige el flujo de procesamiento desde un registro acumulador o equivalente. Con lo anterior en mente, se puede describir una "computadora de côdigo hilvanado directo" como una mâquina en la cual una "instrucciôn" controla el PC (registro contador de programa), o el registro de posiciôn. El côdigo generado para êsta mâquina consistirâ de una lista lîneal de direcciones de rutinas a ser ejecutadas.

3.1.1.1 Côdigo hilvanado directo : Hardware y Software

Imaginemos una computadora que trabaja de la manera siguiente:

Paso 1. S, el valor de la palabra de memoria a la cual apunta el registro contador de programa, se trae a la unidad de procesamiento.

Paso 2. (a) La rutina que inicia en la localidad S de memoria se ejecuta.

Paso 2. (b) El valor de el contador de programa se incrementa en uno.

Paso 3. Continua en el Paso 1.

Esta mâquina puede ser llamada una computadora de côdigo hilvanado directo.

Es factible el construir un dispositivo fîsico que corresponda con la descripciôn anterior. Sin embargo, esto es innecesario. Se demostrarâ que es posible el transformar econômicamente una computadora de propôsito general en una computadora de côdigo hilvanado directo usando un programa.

Describiremos la implantaciôn en un microcomputadora basada en un Z-80. El contador de programa corresponderâ al registro IY del Z-80. Entonces para usar la "micro" como una computadora de côdigo hilvanado solo se necesita que cada una del conjunto de rutinas descritas en el Paso 2 (a) termine con las instrucciones:

```
INC IY
INC IY                                  ; Paso 2 (b)
JMP (IY)                                ; Pasos: 1 y  2 (a
```

las cuales ligan el fin de una rutina con el inicio de la siguiente. Esta es la notaciôn en ensamblador de Z-80 para:

Paso A. Incrementa el valor del registro IY, tal que apunte a la siguiente instruciôn e j e c u t a b l e.

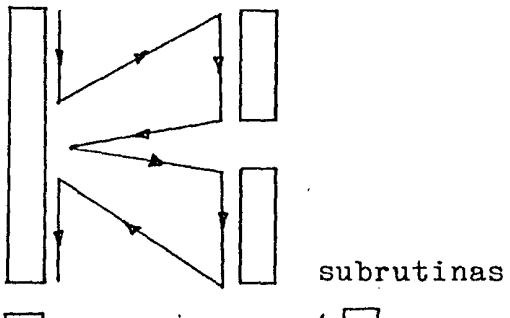
Paso B. Transfiere el control a la rutina que inicia en la localidad cuya direcciôn es el valor del registro IY.

Debe notarse que el paso B hace exactamente lo que se especifica en los Pasos 1 y 2 (a) de nuestra descripciôn de la mâquina de côdigo hilvanado. Similarmente el paso A hace exactamente lo que se especificô en el paso 2 (b). Hemos de hecho transformado una micro-computadora de propôsito general en una micro-computadora de côdigo hilvanado.

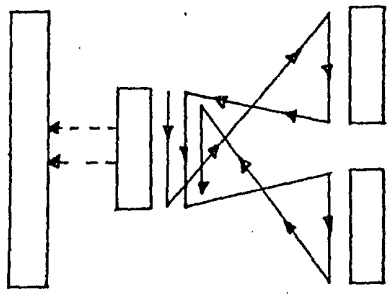
Si una computadora contiene instrucciones que puedan incrementar un registro y puedan cargar el contador de programa a través de dos niveles de indirecciôn o su equivalente, entonces tal procedimiento es posible. Si todo esto puede hacerse en una sola instrucciôn, por ej. en el caso de la mini-computadora PDP-11 con la instrucciôn:
JMP @ (R) + donde R es uno de los registros de propôsito general, entonces se puede lograr una implantaciôn muy econômica usando dicha instrucciôn.

Lo que se ha creado de hecho es côdigo de intêrprete que no necesita un programa que lo interprete. Las figuras siguientes muestran el flujo de control para los diferentes tipos de côdigo descritos:

flujo de control para côdigo de mâquina:



flujo de control para côdigo de intêrprete:



flujo de control para côdigo hilvanado directo:

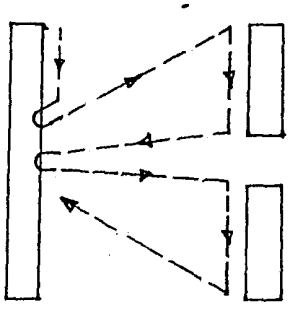


Fig. 3.1 Flujo de control para diferentes tipos de côdigo (tomado de [Bell 73])

De lo anterior se puede concluir que los programas traducidos a côdigo hilvanado se ejecutan mâs râpidamente que los interpretados convencionalmente porque:

- se elimina la llamada al côdigo de control del intêrprete.
- desaparece del ciclo principal de la mâquina (traer la instrucciôn, decodificarla , brincar al subrutina asociada a ella) la parte de decodificaciôn.

En lo que se refiere a el espacio de memoria que se ocupa, es de esperarse que sea mayor que el de côdigo intêrpretado ya que el campo de operaciôn crece porque ahora es una direcciôn de memoria. Sin embargo, aquî se obtiene una ventaja adicional ya que al no existir la tabla que relaciona los côdigos de operaciôn con direcciones de subrutinas, el proceso de añadir nuevas subrutinas al intêrprete se fâcilita grandemente.

Se debe notar que la comparaciôn hecha excluye la posibilidad de tener un procesador cuyas instrucciones tengan una correspondencia uno a uno con las instrucciones de un lenguaje de alto nivel.

3.1.2 Côdigo hilvanado indirecto

El côdigo hilvanado indirecto es una forma eficiente de côdigo para ser intêrpretado. Requiere menos espacio que el côdigo hilvanado directo y permite implantaciones independientes de la mâquina que lo ejecutarâ. A continuaciôn se establece la diferencia entre estos dos tipos de côdigo:

El Côdigo Hilvanado Directo involucra la generaciôn de côdigo que consiste de una lista lineal de direcciones de rutinas a ejecutarse. Algunas de estas rutinas son de "biblioteca", p.ej. la rutina para sumar dos enteros. En cambio las rutinas para accesar los operandos son especîficas a un programa dado y deben ser generadas como parte de la salida del compilador. En la siguiente figura se muestra un ejemplo:

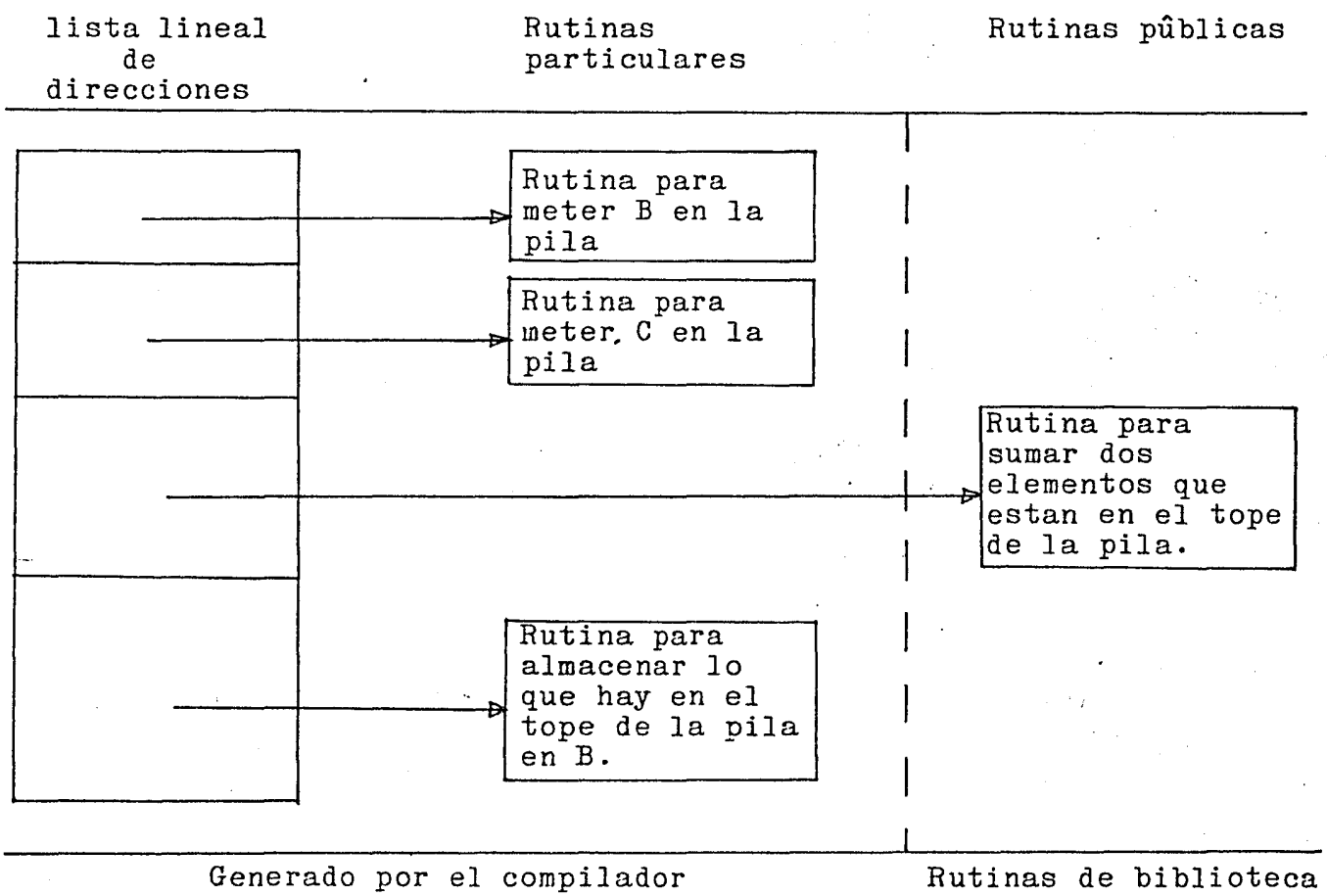


Fig. 3.2 Côdigo hilvanado directo para: B := B + C
(tomado de [Dew 75])

El côdigo hilvanado indirecto consiste de una lista lîneal de direcciones de localidades que contienen las direcciones de las rutinas a ser ejecutadas. Este nivel de indirecciôn se ilustra con el ejemplo que se muestra a continuaciôn:

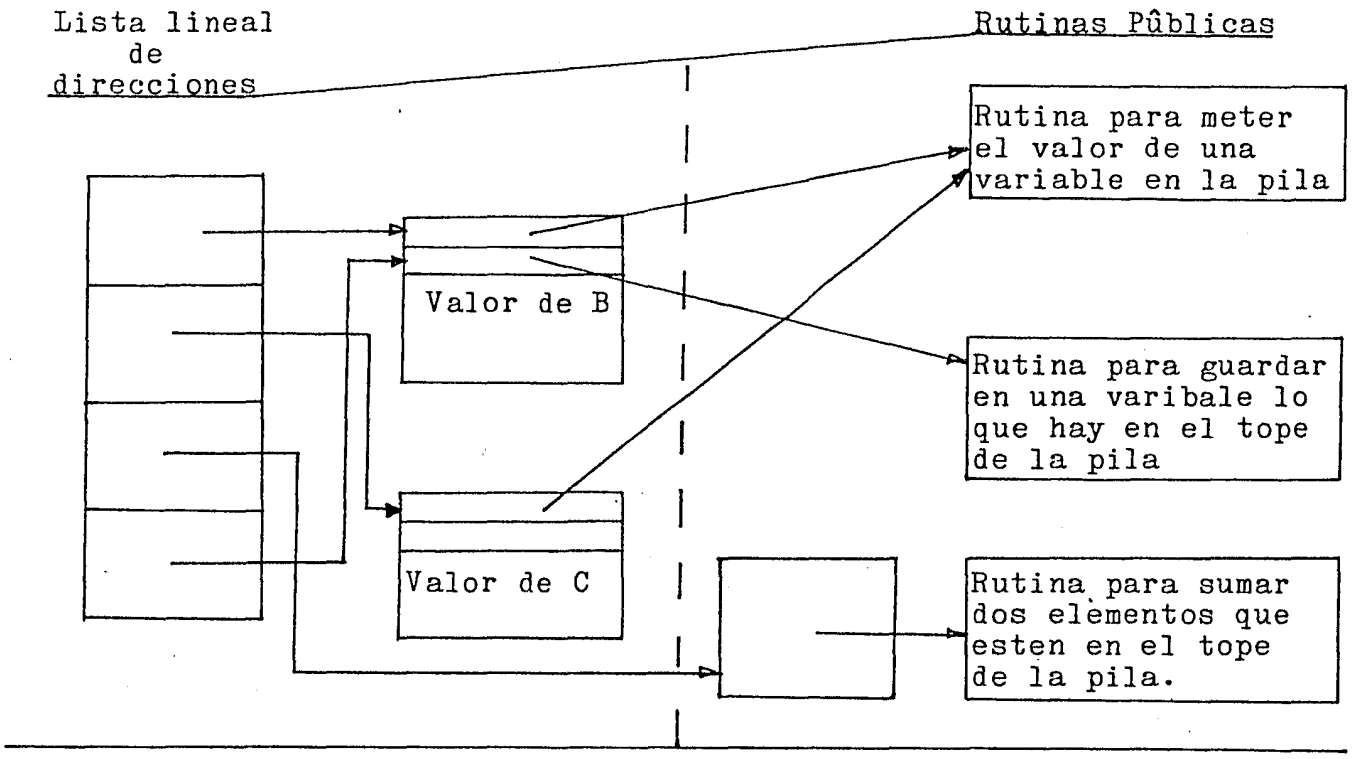


Fig. 3.3 Côdigo hilvanado indirecto para: B := B + C
(tomado de [Dew 75])

Comparando las dos figuras anteriores se nota una importante diferencia cualitativa. El côdigo hilvanado indirecto no involucra la compilaciôn de ningûn côdigo como tal, s o l o d i r e c c i o n e s . En base a que todas las mâquinas implementan el concepto de "palabra", locadidad que puede contener direcciones, el côdigo hilvanado indirecto puede ser generado en una forma esencialmente independiente de la mâquina que lo ejecutarâ.

En el caso del côdigo hilvanado directo, se involucra la generaciôn de rutinas que accesen operandos lo cual hace que el lograr independencia de la mâquina sea mucho mâs dificil. Otro aspecto que es importante recalcar es el hecho de que escogiendo de manera adecuada el côdigo que se ejecuta al llamar a una rutina, êsta puede comportarse como una subrutina o como una varible.

Para examinar los requerimientos de espacio y tiempo se compararan los côdigos que se generarîan en ambos casos, usando el ensamblador para el Z-80:

Côdigo Hilvanado Directo:

```
dir1      ;  push B
dir2      ;  push C
dirsum    ;  add
```

```
dir1: ld    hl,(BVAR)
      push  hl
      inc   iy
      inc   iy
      jp    (iy)
dir2: ld    hl,(CVAR)
      push  hl
      inc   iy
      inc   iy
      jp    (iy)

BVAR: defb  0       ; valor de B
      defb  0
CVAR: defb  0       ; valor de C
      defb  0
```

```
dirsum: pop   hl
        pop   de
        add   hl,de
        push  hl
        inc   iy
        inc   iy
        jp    (iy)
```

Côdigo Generado

Rutinas de Biblioteca

Espacio usado por el côdigo generado: 24 bytes
Nûmero de ciclos T requeridos para su ejecuciôn: 180

Côdigo Hilvanado Indirecto:

```
        BVAR    ; push B
        CVAR    ; push C
        dirsum  ; add

        0       ; valor de B
        0
BVAR:   pushvar
        0       ; valor de C
        0
CVAR:   pushvar
```

```
pushvar:  ld    h,(ix-1)
          ld    l,(ix-2)
          push  hl
          inc   ix
          inc   ix
          ld    l,(ix)
          ld    h,(ix+1)
          jp    (hl)
```

```
dirsum:   $+2
          pop   hl
          pop   de
          add   hl,de
          push  hl
          inc   ix
          inc   ix
          ld    l,(ix)
          ld    h,(ix+1)
          jp    (hl)
```

Côdigo Generado Rutinas de Biblioteca

Espacio usado por el côdigo generado: 14 bytes
Nûmero de ciclos T requeridos para su ejecuciôn: 225

Se puede decir que se sacrifica râpidez de ejecuciôn a cambio de ahorrar espacio y lograr una representaciôn homogênea del côdigo generado. Esto redunda en un mayor grado de portabilidad, y extensibilidad manteniendo un buen nivel de eficiencia. La combinaciôn de estas propiedades y el uso de conceptos como: pila y notaciôn polaca postfija llevan el sello de la simplicidad y permiten la creaciôn de sistemas que no estan a nivel de progamaciôn en lenguaje de mâquina, ni en el de programaciôn con lenguajes de muy alto nivel. En su lugar representan un puente entre estos dos estadîos y le dan al programador las caracterîsticas y flexibilidad de alguno de ellos o de los dos, en la combinaciôn que mejor se adecuê a la aplicaciôn.

La implantaciôn de côdigo hilvanado indirecto en una microcomputadora de 8 bits, como el Z-80, tiene como costo una reducciôn en la eficiencia del intêrprete. Debido por un lado a que no existen instrucciones de mâquina con "post-incremento", es decir que tenga como efecto secundario un incremento del registro usado, y por otro a las limitaciones del lenguaje de mâquina para direccionar palabras de 16 bits. Esta reducciôn de la eficiencia del intêrprete se manifiesta en el espacio que ocupa, lo cual hace imprâctico el duplicarlo al final de cada subrutina. Por ejemplo en el caso de la rutina "pushvar" el interprête ocupa el 55.8 % del total de espacio que ocupa la rutina.

Una implantaciôn que toma en cuenta lo anterior es la que propone [Loe 81]. Loelinger sugiere para implantar en un Z80 un intêrprete de côdigo hilvanado indirecto usando el registro BC como contador de programa y el registro DE como apuntador al prôlogo (côdigo inicial de cada subrutina). A este intêrprete se le denomina intêrprete interno o intêrprete de direcciones.

intêrprete interno:

```
    ld      a,(bc)
    ld      l,a
    inc     bc
    ld      a,(bc)
    ld      h,a
    inc     bc              ;  1er nivel de indirecciôn.
    ld      e,(hl)
    inc     (hl)
    ld      d,(hl)
    inc     hl
    ex      de,hl
    jp      (hl)            ;  2o nivel de indirecciôn.
```

Usando la estructura tradicional de guardar una sola copia del côdigo de control del intêrprete, al têrmino de cada subrutina se deberâ, entonces poner una instrucciôn que brinque al intêrprete (interno). A pesar de esto, seguirân siendo vâlidas las ventajas antes proclamadas: facilidad de agregar subrutinas al intêrprete, ausencia de una tabla que traduzca côdigos de operaciôn a direcciones, homogeneidad de procesamiento para subrutinas y tipos de datos, y un intêrprete sencillo. Si en el caso particular del Z-80 se almacena la direcciôn del intêrprete en el registro IY, el brinco es bastante râpido.

Para tener la posibilidad de anidar rutinas dentro de otras, se usa una pila para mantener la direccion del contador de programa BC, y se le asigna la nueva direcciôn de la lista "anidada". A esta se le añade al final una rutima que recupera de la pila el valor anterior del contador de programa que corresponde a la rutina que hizo la llamada. A esta pila generalmente se le llama: pila de retorno. Loelinger escoge IX como apuntador a la pila de retorno.

```
dec     ix
ld      (ix),b
dec     ix
ld      (ix),c
ld      c,e                     ; push  pc  actual
ld      b,d                     ; pc <- dir nueva lista
jp      (iy)                    ; salta al intêrprete
```

y el côdigo de regreso de las rutinas anidadas serâ:

```
defw    $+2
ld      c,(ix)
inc     ix
ld      b,(ix)
inc     ix                      ; pop pc de la rutina que llamô
jp      (iy)                    ; salta al intêrprete
```

Esta têcnica invierte la posiciôn de la informaciôn que le dice a la mâquina que estâ entrando a una rutina nueva. En los intêrpretes convencionales, los côdigos de operaciôn asociados con las direcciones son interpretados a n t e s de entrar a la nueva rutina. En este caso, el côdigo del prôlogo sirve para la misma funciôn, pero es invocado d e s p u ê s de que la direcciôn de la nueva lista ha sido usada.

Lo se ha definido es, en efecto, un mecânismo para que la rutina que ha sido llamada determine que clase de rutina es y como debe salvar la informaciôn de retorno; en contraste, el procedimiento clâsico demanda que la rutina que hace la llamada genere esta informaciôn. En cierto sentido, la mâquina sabrâ a que clase de rutina es a la que esta entrando hasta que este allî. Êsta capacidad de "auto-definiciôn" es la piedra angular para muchas de las capacidades que se discuten mâs adelante.

Los artículos de [Bel 73] y [Dew 75] son referencias indispensables para aquêl que se interese por el côdigo hilvanado y los intêrpretes.

3.2 Forth

Forth fuê inventado a finales de los 60ˆs por Charles Moore, quiên estaba tratando de elevar su propia productividad como "programador de aplicaciones", ya que el reto impuesto por el desarrollo de las minicomputadoras encontrô una respuesta desalentadora por el lado del software en la industria de la computaciôn. Mucha de la programaciôn de minicomputadoras se haĉia en ensamblador; El desarrollo de lenguajes de alto nivel habîa consistido en adaptar lenguajes desarrollados para computadoras grandes (notablemente FORTRAN y BASIC) en la pequeñas memorias y pequeñas longitudes de palabra que caracterizan a las minicomputadoras. Similarmente, los sitemas operativos de las minicomputadoras han imitado los principios estructurales de los sistemas grandes orientados al procesamiento fuera de lînea. Todo esto diô como resultado, no solo un uso sub-ôptimo del hardware de las minicomputadoras, sino que tambiên se tienen compiladores y sistemas operativos cuyo uso es difîcil y complejo, especialmente para los trabajos que se realizan en lînea, objetivo que la mayorîa de las minicomputadoras pretende lograr.

El objetivo de Moore era, de hecho, el invertir el sentido de este avance y sacar un mayor provecho de la economîa del tamaño, costo y eficiencia que caracterizan a la minicomputadora. Esto requiere que la tarea que tiene el programador de usar eficientemente el hardware de la mâquina se facilite. Forht es el sistema que obtuvo como resultado.

Forht fuê descrito por êl como " Un sistema que que incluye en una sola estructura integrada un lenguaje de alto nivel, un macro ensamblador, un sistema operativo para multiprogramaciôn y utilerîas, ademas de otros conceptos, como notaciôn polaca postfija, organizaciôn orientada a pilas y memoria virtual para el almacenamiento de datos. El uso de estas têcnicas no es exclusivo de Forht, sin embargo su uso trae buenas consecuencias en este caso ya que el uso de rutinas re-entrantes es, por ejemplo, totalmente natural al escribir en Forth. Como lenguaje Forth permite el uso de estructuras lôgicas que son completamente consistentes con los conceptos de programaciôn estructurada."

Actualmente las microcomputadoras estân pasando por el mismo fenômeno que reconociô Moore. Y a pesar de que Forth no fuê diseñado para este tipo de mâquinas, es una alternativa que explota, a un nivel aceptable de eficiencia, el hardware de estâs mâquinas. Para los programadores de sistemas de este tipo de mâquinas Forht permite, al usar una sola estructura de lenguaje, el salvar la distancia que existe entre algûn tipo de hardware y la aplicaciôn. Como lo muestra la fig. siguiente.

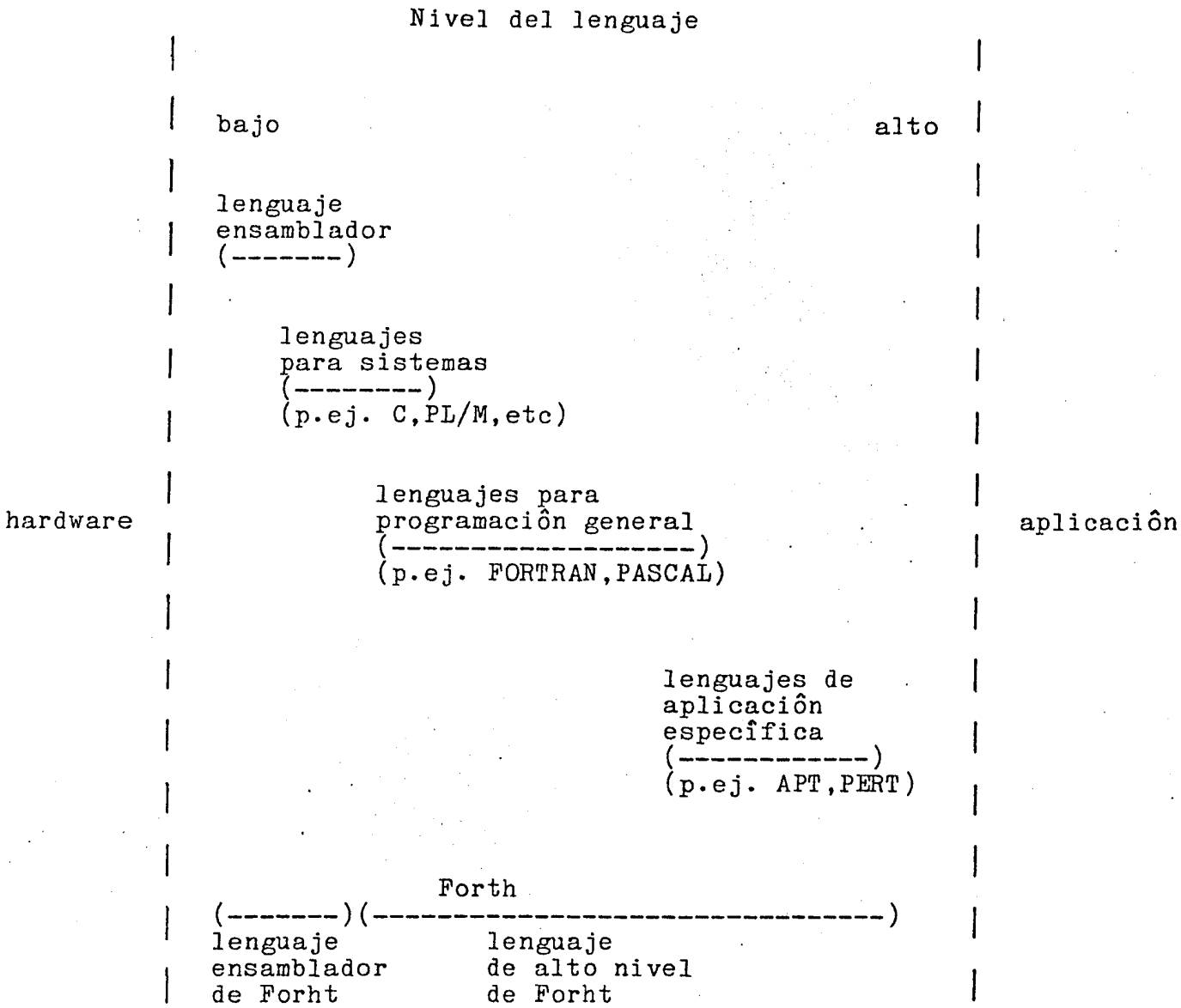


Fig. 3.4 Espacio a salvar por el software (tomado de [Har 81])

Forth estâ basado en una mâquina de côdigo hilvanado. En la secciôn anterior se describiô una caracterîstica del côdigo hilvanado: la representaciôn homogênea entre subrutinas (funciones) y tipos de datos. La diferencia entre los tipos de entidades en un ambiente hilvanado radica en sus prôlogos, que contienen las instrucciones de como procesar cada entidad. Los prôlogos indican si las entidades deben interpretarse como côdigo o como variables o como arreglos o cualquier tipo de dato imaginable.

La mâquina virtual que ejecuta el côdigo se denomina intêrprete interno, es en extremo sencilla (ver la secciôn anterior) y en algunos lenguajes de mâquina ocupa dos instrucciones. No obstante, el intêrprete interno presupone que el prôlogo no està escrito en côdigo hilvanado sino en lenguaje de mâquina. A pesar de ello, Forth atinadamente provee un prôlogo especial que hace que se ejecute en côdigo hilvanado un segundo prôlogo. Esto permite crear nuevas entidades cuyo prôlogo està escrito en Forth. De esta manera se establecen dos tipos de funciones: primitivas y secundarias. Las primitivas serân todas aquellas funciones cuyo prôlogo este escrito en lenguaje de mâquina, las secundarias serân las que como prôlogo tengan uno que debe ser ejecutado como côdigo hilvanado. Como consecuencia, Forth habilita al usuario para que defina el côdigo que se ejecuta al hacer referencia a algûn tipo de dato.

3.2.1 Extensibilidad

El usuario tiene acceso a todas las variables del intêrprete de manera que es fâcil extenderlo, definir nuevas estructuras de control y crear tipos de datos ausentes en el sistema original. Ademâs, debido a que el usuario tiene acceso al diccionario de entidades (descrito mâs adelante), Forth es extensible a dos niveles en los que la mayorîa de los lenguajes se comportan como cajas negras. Los niveles a los que Forth se puede extender son:

- Definiciôn del côdigo de nuevas funciones.

  Este nivel de extensibilidad es comûn a casi todos los lenguajes y no representa ninguna innovaciôn. Probablemente el ûnico aspecto interesante en este sentido es que Forht permite que una funciôn regrese varios valores, es decir tenga diversos parâmetros de salida.

- Definiciôn del côdigo de creaciôn de los tipos de datos.

  Al tener acceso al diccionario, el usuario està facultado para asignar un valor inicial a los tipos de datos. Ciertos lenguajes de programaciôn contienen instrucciones para iniciar el valor de las variables, pero debido a que en la mayorîa de ellos existe un conjunto predefinido de tipos de datos (digamos arreglos y estructuras), no hay manera de establecer el côdigo de creaciôn de los datos en el lenguaje mismo.

  El programador de Forth puede no solo crear funciones, sino crear funciones que creen funciones.

- Definiciôn del côdigo de ejecuciôn de los tipos de datos.

  Equivale a que es factible programar los prôlogos de las entidades. Este nivel de extensibilidad està muy relacionado con el anterior; de hecho, al definir una funciôn que crea funciones, es necesario programar tanto el côdigo de creaciôn como el prôlogo del tipo de dato, dentro de la misma entidad.

3.2.2 Parâmetros

Forht emplea dos pilas: una para los parâmetros de las subrutinas y otra para las direcciones a las que debe regresar al terminar de procesar las funciones que han sido llamadas. La mayorîa de las implantaciones de los lenguajes derivados de Algol emplean un mecanismo para llamadas a subrutinas que utiliza una sola pila, en la que almacenan tanto los parâmetros como las direcciones de retorno. Como consecuencia al llamar una subrutina cuyos parâmetros han sido previamente colocados en la pila, se almacena todavîa arriba de los parâmetros el valor del contador de programa. La subrutina llamada, a su vez, tiene que "destapar" la pila extrayendo el contador de programa antiguo para sacar los parâmetros, y volver a tapar la pila con objeto de que al regresar se tome el valor adecuado del contador de programa.

La existencia de dos pilas separadas simplifica el paso de parâmetros e incrementa la velocidad de llamado de subrutinas.

Los parâmetros de Forth no se pasan ni por valor ni por referencia. Al llamar a una funciôn, êsta extrae sus parâmetros de la pila tal y como fueron dejados por la funciôn que la llamô. La funciôn que llama simplemente coloca valores en la pila, pero depende del prôlogo del tipo de dato al que haga referencia para poder decir si en la pila queda la direcciôn o el valor de una variable.

En caso de que se trate del tipo de dato v a r i a b l e , que es variable escalar global y tiene como prôlogo regresar en la pila la direcciôn de la variable, parecerîa que se estâ haciendo una llamada por referencia. No obstante, existe la funciôn "@" (arroba) que permite calcular el valor de la variable antes de llamar a la subrutina, lo que equivale a una llamada por valor.

Ademâs el tipo de dato c o n s t a n t automâticamente regresa el valor del dato en la pila sin necesidad de hacer una indirecciôn. Este tipo de dato difiere de lo que la mayorîa de los lenguajes entienden por "constante", es decir, un valor conocido durante compilaciôn e imposible de modificar en el programa objeto sin compilarlo de nuevo. El tipo de dato c o n s t a n t se comporta mâs bien como una variable que regresa su valor en la pila y aunque no existe una forma automâtica de hacerlo, sî es factible modificar su valor durante la ejecuciôn del programa, si se averigua su direcciôn en memoria.

Un tipo de dato que se agregô a Forth, y que por lo tanto no es estandar, es el denominado g l o b a l. Este tipo de dato se puede ver como el tipo c o n s t a n t , excepto que permite que un conjunto restringido de funciones conozcan la direcciôn de la variable creada y por lo tanto la puedan modificar.

En resumen, el hecho de llamar a una subrutina con parâmetros globales no implica que se llamen por valor o por referencia, mâs bien depende del prôlogo de los parâmetros; para el tipo de dato v a r i a b l e se sobreentiende una llamada por referencia a menos que el usuario indique lo contrario, y para los tipos de dato c o n s t a n t y g l o b a l es al revês.

De manera similar, el usuario puede definir tipos de datos que regresen varios valores en la pila, simulando una llamada por valor a un arreglo, por ejemplo.

Las variables locales y los parâmetros de una funciôn no tienen nombre. El usuario debe hacer referencia a ellos por el lugar que ocupan en la pila con respecto al tope. Consecuentemente, dentro de una funciôn es posible que a la misma variable se haga referencia de mâs de una forma distinta, dependiendo del nûmero de datos que exista sobre la variable en la pila. Este es probablemente el "defecto" mâs notorio en Forth. Y ya que las rûtinas de Forth se comunican entre ellas, en gran medida, por medio de la pila de parâmetros, el programador o el lector de un programa tiene que crear una imagen mental de la pila y sus cambios durante la ejecuciôn para comprender su comportamiento, con lo cual un programa escrito en Forth tiende a ser crîptico en su apariencia.

Aunque Forth es suficientemente flexible como para construir un compilador con Forth mismo, que sî reconozca nombres de parâmetros y variables locales no se sabe de ningûn intento en este sentido.

Si vemos a la asignaciôn como la llamada a una subrutina, podemos afirmar que la mayor parte de los lenguajes de programaciôn que preveen la asignaciôn de variables, presuponen que la variable que va a ser modificada es llamada por referencia y que todas las variables presentes en la expresiôn cuyo valor debe recibir la variable son llamadas por nombre. En Forth no existe esta presuposiciôn; la asignaciôn "!", al igual que todas las funciones, toma sus parâmetros tal y como fueron colocados por la funciôn que la llamô, de manera que es labor del usuario el cuidar de colocar el valor como primer parâmetro de la asignaciôn y la direcciôn de la variable como segundo.

3.2.3 Intêrprete externo

El intêrprete de comandos recibe el nombre de intêrprete externo. Emplea notaciôn polaca postfija y evalûa expresiones con una mâquina virtual (de mâs alto nivel que el intêrprete externo) que

- si encuentra un nûmero lo convierte a binario y lo introduce en la pila
- si no es nûmero busca en el diccionario suponiendo que se trata de una funciôn; si la encuentra la ejecuta sin verificar que se haya pasado el nûmero adecuado de parâmetros, y si no, marca error.

Los ûnicos delimitadores son el espacio en blanco y el retorno de carro. Esto hace que los nombres de las funciones puedan contener cualquier carâcter (diferente de los anteriores) como parte de su nombre.

Cada vez que termina de ejecutar las funciones que escribiô el usuario en un renglôn, el intêrprete externo examina que no se hayan extraido mâs parâmetros de los necesarios, en cuyo caso marca error.

El intêrprete externo solamente introduce en la pila los nûmeros pero nunca las cadenas de caracteres. No es posible pasar como parâmetros, en forma tan directa como los nûmeros, caracteres alfanumêricos. Forth emplea notaciôn polaca prefija para parâmetros alfanumêricos. De manera que las funciones deben buscar sus parâmetros alfanumêricos en el arreglo (buffer) asociado a la terminal, al cual tienen acceso. Sin embargo, existen variantes de Forth, como Stoic, a cuyos parâmetros alfanumêricos se les antepone un apôstrofe y conservan la notaciôn polaca postfija.

3.2.4 Compilador

El llamado "compilador" de Forth es muy primitivo como para que amerite ese nombre; mâs bien se parece a la funciôn READ de Lisp, pues bâsicamente traduce identificadores externos de funciones (sus nombres) a identificadores internos (sus direcciones) con objeto de no tener que hacerlo durante la ejecuciôn. Esta simplicidad es el resultado de implantar Forth como un intêrprete de côdigo hilvanado. En realidad existen varios compiladores dependiendo del prôlogo que asignan a la funciôn y de como traducen su c u e r p o. El que se usa con mâs frecuencia es ":" que sirve para crear funciones secundarias. Este compilador, una vez que ha incluido a la nueva entidad en el diccionario y establecido su prôlogo como el de una funciôn secundaria (que se vio en la secciôn anterior), efectûa un proceso similar al del intêrprete externo:

- si encuentra un nûmero, incluye en el cuerpo de la funciôn un apuntador a la funciôn "literal" y a continuaciôn el nûmero en binario. El côdigo de la funciôn literal es, como es de esperarse, transferir a la pila el nûmero al que apunta el contador de programa del intêrprete interno y avanzar el contador de programa a fin de que el nûmero no se interprete como una funciôn. (ver el ejemplo de pushvar en la secciôn anterior)

- si encuentra una funciôn, se da uno de dos casos: si se trata de una funciôn "mediata", como son la mayorîa de las funciones de Forth, la busca en el diccionario e incluye su direcciôn en el cuerpo de la funciôn que estâ siendo definida. Si resulta ser una funciôn "inmediata", la ejecuta. Un primer ejemplo de funciones inmediatas es la que termina la definiciôn de las funciones ";". Si esta funciôn no se ejecutara durante la compilaciôn, no habrîa manera de finalizar el cuerpo de las funciones secundarias. Entre otras labores, esta funciôn debe incluir un apuntador al intêrprete interno como ûltima instrucciôn de la nueva entidad y regresar al intêrprete externo.

Otras funciones que se ejecutan durante la compilaciôn son DO, LOOP, IF, ELSE y THEN. DO es una funciôn que debe ir siempre acompañada por LOOP, que una vez compilada, toma de la pila dos valores entre los cuales hace variar de uno en uno un îndice (equivale al FOR de Algol) y entre cada variaciôn ejecuta el côdigo localizado entre DO y LOOP. LOOP debe traducirse entonces en un apuntador a cierta funciôn que decide si el côdigo debe repetirse una vez mâs e incrementar el valor del îndice o si el flujo de control debe seguir adelante.
Sin embargo la longitud de ese brinco hacia atrâs se calcula durante la compilaciôn de la siguiente manera: DO mete en la pila de datos la direcciôn en la que apareciô el mismo. LOOP, a su vez, toma de la pila la direcciôn dejada por DO y la resta de s u direccion, traduciendo el LOOP por un brinco condicional relativo.

En forma similar se compilan IF y THEN. El IF, una vez compilado toma de la pila un valor. Si es cero, brinca a la instrucciôn inmediatamente despuês del THEN y si es distinto de cero, ejecuta el côdigo entre el IF y el THEN. Al igual que en el caso anterior, la longitud del brinco se calcula durante la compilaciôn. Cuando el compilador encuentra un IF, lo traduce a un apuntador a otra funciôn, digamos 0-BRANCH, la cual tiene como côdigo lo que se espera del IF una vez compilada la funciôn, y mete a la pila la informaciôn adecuada para que el THEN calcule el tamaño del brinco y llene la palabra de memoria que esta a continuaciôn de 0-BRANCH. El THEN desaparece despuês de la compilaciôn. Ademâs del IF ... THEN , tambiên existe la secuencia de instrucciones
IF ... ELSE ... THEN. (Notese que el THEN aquî juega el papel del ENDIF de PASCAL).

Como puede apreciarse Forth esta diseñado para la programaciôn estructurada, eliminando los GO TOs. Asimismo, es factible anidar estructuras de control, pues se comunican a travês de una pila en la que extraen primero los datos asociados a las estructuras mâs interiores.

Forth no verifica que todos los THEN casen con los IF, ni los DO con los LOOP, por lo que es comûn que aparezcan errores de programaciôn originados por falta de "terminadores" de estructuras de control. Aun asî, observamos que siempre entre la definiciôn de una funciôn y la siguiente la pila de datos debe estar vacîa, es fâcil hacer que la funciôn ";" verifique esta condiciôn y marque error en caso de que no se cumpla. Esta têcnica no detectarâ exactamente en que parte de el programa estâ el error, pero sî en cual funciôn.

La estructura de una funciôn creada con el compilador ":" es la siguiente:

: nombre-de-la-fiunciôn côdigo ;

A la implantaciôn de Forth que se tiene se le agregaron variosn compiladores con el fin de facilitar la tarea de programaciôn en este lenguaje:

":" Este compilador incluye todas las caracterîsticas del ":" permitiendo ademâs el manejo de parâmetros por nombre. Estos parâmetros pueden ser: de entrada,salida,locales o de entrada/salida. Dentro del côdigo los parâmetros se referencian de dos maneras:

- nombre del parâmetro. Se mete en la pila el valor del parâmetro.

- nombre del parâmetro precedido por un punto y sin espacio entre el punto y el nombre. Se asigna el valor que hay en la pila a la variable.

La estructura de un funciôn creada con el compilador "::" es la siguiente:

:: nombre-de-la-funcion parametros & codigo ;

Los parâmetros van antepuestos a una llamada a ">>" si son de entrada, "<<" si son de salida, "><" si son tanto de entrada como de salida y de "**" si son locales.

"*:" Este es un depurador de ":". Para efectos semânticos funciona en forma idêntica al compilador de ":". Sirve para depurar porque inserta una llamada a un intêrprete externo antes de cada llamada a una funciôn mediata. Este intêrprete suspende la ejecuciôn del programa e imprime el nombre de la funciôn que va a ejecutar. La ejecuciôn prosigue de las siguientes formas:

- <ctrl> C continûa la ejecuciôn hasta la siguiente funciôn mediata.

- <ctrl> F continûa la ejecuciôn hasta el final de la funciôn que se esta depurando.

- abort en el caso que se quiera abortar la depuraciôn.

- q en el caso que se quiera abandonar el proceso de depuraciôn.

"*::" Es el depurador de "::".

"m:" Crea un funciôn especializada en modificar determinada va-
variable creada con el compilador "global". Al entrar al
côdigo, automâticamente se inserta en la pila la direc -
ciôn de la variable. La estructura de la funciôn serâ:

```
m:  nombre-de-la-variable-global   nombre-de-la-funciôn   côdigo  ;
```

"m::" Tiene la misma funciôn que "m:", pero al igual que el
compilador "::" reconoce parâmetros. La lista de parâmetros
debe contener por lo menos un parâmetro de entrada o de
entrada-salida, que es la direcciôn de el tipo de dato global
al que està asociada esta funciôn. Este parâmetro debe ser
el ûltimo en caso de que haya varios (se entiende que sean de
entrada o de entrada-salida).

Su estructura es la siguiente:

```
m:: nombre-de-la-variable-global  nombre-de-la-funcion

    parametros  &  codigo  ;
```

"*m:" Es el depurador de "m:" .

"*m::" Es el depurador de "m::" .

Existen otros compiladores como CREATE que genera funciones primitivas ,es decir en côdigo de mâquina: Z-80, asi como VARIABLE, CONSTANT y GLOBAL que crean variables, constantes. Para crear, por ejemplo, una variable se usa la forma <valor inicial> VARIABLE <nombre>. (conserva la notaciôn postfija para valores numêricos y la prefija para cadenas de caracteres.) Para constantes y globales se usa la misma estructura cambiando tan solo VARIABLE por CONSTANT o GLOBAL segûn el caso.

3.2.5 Diccionario

Las entidades de Forth se almacenan en un diccionario. Para propôsitos de su uso, es irrelevante su estructura, pero vale la pena mencionar algunas caracterîsticas que tienen consecuencias importantes. La mayor parte de las implantaciones de Forth emplean una lista ligada de funciones por lo que tanto el compilador como el intêrprete externo deben realizar una bûsqueda secuencial. Aunque algunas variantes de Forth, como IPS, utilizan una funciôn de dispersiôn (hashing), la lista ligada permite implantar con facilidad el concepto de "vocabularios".

Los vocabularios son conjuntos de funciones asociados a un nombre. Con ellos, es posible ocultar al compilador y al intêrprete externo grupos de funciones permitiendo al usuario emplear el mismo nombre de funciones bajo contextos distintos. Los conjuntos no necesitan forzosamente ser disjuntos. En Forth es factible incluir digamos las funciones bâsicas en todos los conjuntos. Por ejemplo, en un sistema con un editor de texto y un ensamblador, el usuario puede utilizar en su programa de apli caciôn nombres de funciones que yazcan en los vocabularios del editor o del ensamblador. Los vocabularios fueron de especial utilidad en este trabajo debido a que se almacenaron las entidades del que permiten la utilizaciôn del editor en el diccionario de Forth y se empleo un vocabulario distinto para el editor de textos.

Los nombres de las entidades se guardan en su totalidad, a diferencia de las implantaciones convencionales en los que solo se almacena los tres primeros caracteres, ademâs del nûmero de caracteres del nombre. A estos bytes, junto con el apuntador a la funciôn anterior se le denomina e n c a b e z a d o de la funciôn.

3.2.6 Conclusiones

Forth ha recibido y con razôn el calificativo de un lenguaje de solo escritura (write only). El lenguaje es tan crîptico como APL, en parte debido a la notaciôn polaca pero sobre todo porque los parâmetros y las variables locales carecen de nombre.

La notaciôn postfija no es una desventaja importante pues es bien sabido que mucho depende de la costumbre del programador. El usuario de Lisp (que emplea notaciôn polaca prefija) râpidamente adquiere habilidad para leer sus programas. La falta de nombres en las variables locales y en los parâmetros sî resulta en un inconveniente grave y es de extrañar que no se haya corregido. El compilador "::" se diseño con el fin de subsanar esta falta, ya que reconoce nombres de parâmetros [Ros 84]. Este compilador realmente permitiô el salvar restricciones ajenas al problema, checar el estado de la pila continuamente y facilitar asî la tarea de programaciôn.

A pesar de que el compilador de Forth realiza una traducciôn muy sencilla, es costumbre que se almacene una copia fuente del programa ademâs del côdigo objeto. Existen "descompiladores" de Forth, pero no son pópulares debido a que normalmente Forht guarda funciones como arreglos. Para insertar instrucciones se necesitarîa recorrer las entidades en la memoria o emplear otra zona de memoria generando basura.

La alternativa de insertar instrucciones no es conveniente porque los identificadores internos de las entidades de Forth son sus d i r e c c i o n e s. Al mover de lugar una funciôn, a pesar de que no afectaría los brincos relativos, sí significa modificar todas las referencias a la funciôn desplazada. Para que este mecanismo que traslada entidades en la memoria fuera prâctico, habría que emplear otro tipo de identificadores internos que no dependiera de la direcciôn en la que se localiza la entidad.

Hay implantaciones de Forth que emplean una indirecciôn mâs en el intêrprete interno para lograr obtener côdigo relocalizable.

El libro de Loelinger [Loe 81] es bueno para la implantaciôn de Forth, pero para una introducciôn menos profunda, consûltense [BYT 80] y [Kog 82], en español [Ger 84a].

4. Implantaciôn

En este capîtulo se describen las estructuras de datos que se utilizarôn en la implantaciôn del editor, asi como las rutinas mâs importantes, en la primera parte. En la segunda se da una descripciôn mâs detallada de las rutinas que componen el editor.

4.1 Estructuras de datos.

4.1.1 Almacên de ediciôn

El almacên de ediciôn se implantô como un arreglo de caracteres. Su funciôn es permitir la ediciôn de un solo renglôn de texto, es decir sobre este arreglo se hacen las modificaciones de la lînea de texto en la que se encuentra el cursor durante la sesiôn. Si la lînea es nueva este arreglo aparece vacîo, si ya existe en la representaciôn interna entonces este arreglo tendrâ una copia de dicha lînea. Este arreglo se accesa por medio de un apuntador, llamado "mx", para borrar y/o insertar caracteres. La posiciôn relativa que tiene mx con respecto al origen del arreglo, es la misma que tiene el cursor con respecto al origen del renglôn sobre el que se encuentra en la pantalla. El movimiento de mx, representa el movimiento que sobre el renglôn realiza el cursor. La estructura de este arreglo se muestra a continuaciôn:

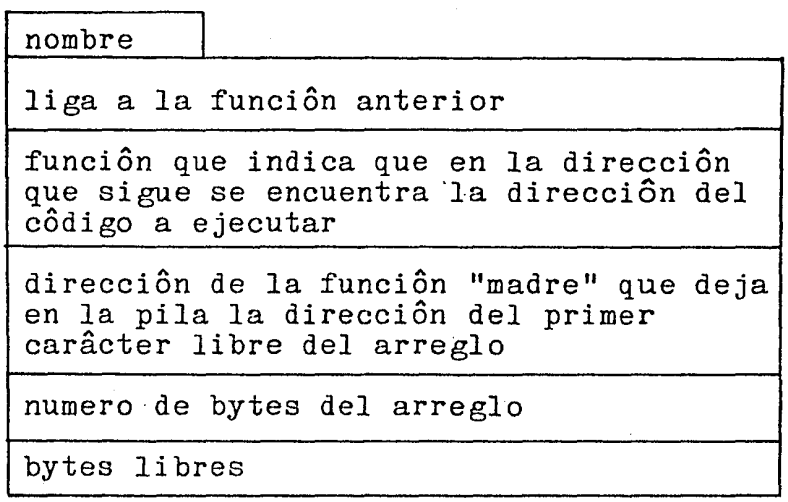


Fig. 4.1 Estructura del almacên de ediciôn.

La implantaciôn de las rutinas que accesan el almacên hacen uso de uno de los niveles mâs altos de extensibilidad de Forth: funciones que crean funciones.

Para la implantaciôn de mx se usô el tipo de dato g l o b a l [Ros 84] con la cual se pretende limitar el acceso a dicho apuntador, la finalidad fuê facilitar el proceso depuraciôn-implantaciôn.

Las rutinas que actuan sobre esta estructura tienen, como funciones mâs importantes, las siguientes:

- Asignar un carâcter en la posiciôn a la que apunta mx, avanzar mx y actualizar la pantalla y la posiciôn del cursor (en la pantalla).
- LLevar registro del status del almacên de ediciôn: Si hubo algûn cambio antes de abandonarlo ; saber la posiciôn de mx ; saber si se esta en modo inserciôn o no.
- Agrandar/Achicar el tamaño del renglôn. (Se usa para insertar o borrar caracteres)
- Bûsqueda/Reemplazo de una cadena de caracteres por otra.
- Transferir el contenido del almacên de ediciôn a la representaciôn interna. (se realiza solo si el status indica que hubo un cambio en el contenido del renglôn).
- Limpiar el almacên de ediciôn. Siempre que se va a editar nuevo se "limpia" el almacên de ediciôn, es decir se llena de caracteres en blanco (carâcter ASCII: 20).

4.1.2 Representaciôn interna del documento.

La representaciôn interna del documento se implantô como una concatenaciôn de renglones (como separador de renglones se usô el carâcter ASCII: OD). Su funciôn es la de permitir la creaciôn, eliminaciôn y ediciôn de renglones de texto. La representaciôn de esta estructura se muestra a continuaciôn

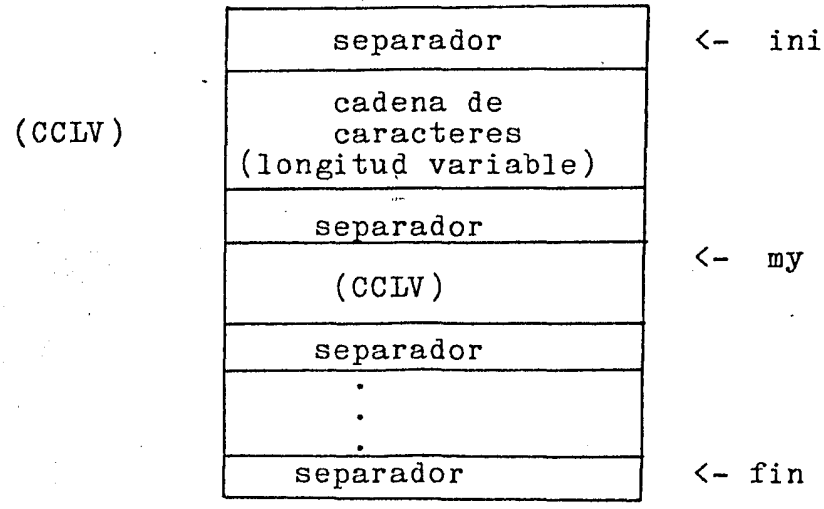


Fig. 4.2 Estructura de la representaciôn interna.

Esta estructura se accesa por medio de un apuntador, "my", el cual apunta a el renglôn en el que esta el cursor durante la sesiôn.

Los lîmites de esta estructura los establecen los apuntadores: "inicio" y "fin", "inicio" se mantiene fijo y "fin" varia su posiciôn de acuerdo a los requerimientos de espaciô del documento. Cuando el documento esta a punto de exceder el lîmite de memoria disponible el usuario recibe un mensaje en la pantalla que le advierte de esto.

Las actualizaciones a esta estructura solo se realizan cuando el cursor abandona el renglôn en el que se encuentra. Esto permite arrepentirse de las modificaciones que se hicierôn al renglôn.

En estrecha realciôn con my trabaja otro apuntador, "cy", el cual se encarga de llevar el registro de la posiciôn del cursor en la pantalla y con êsta informaciôn se sabe si es necesario actualizar el contenido de lo que hay en pantalla, tomando como unidad el renglôn. Esto permite dar la impresiôn de que se recorre el documento hacia el inicio o el fin de êste tomando como "ventana" de enfoque a la pantalla.

La representaciôn interna del documento ocupa lugares diferentes de memoria durante la sesiôn y despuês de ella, con el fin de permitir la compilaciôn incremental y el uso de los recursos de la mâquina en medio de una sesiôn. Utiliza la parte baja de memoria cercana a la ûltima funciôn del diccionario de funciones durante la sesiôn. Fuera de sesiôn se situa en la parte alta de memoria con el fin de dejar espacio libre para que el diccionario crezca al compilar lo editado y/o que el usuario haga uso de los recursos de Forth.

Todos los apuntadores mencionados: "my", "inicio","fin" y "cy" se implantarôn usando el tipo de dato g l o b a l. Lo cual implica que solo cierto tipo de rutinas pueden alterar su contenido.

Las rutinas que actuan sobre esta estructura tienen entre sus tareas mâs importantes:

- Transferir el contenido del renglôn al que apunta my al almacên de ediciôn e inicializar el status de êste. (No hay cambio en el contenido y la longitud, del renglôn actual, por ejemplo).

- Mover el apuntador my un nûmero determinado de renglones hacia el inicio o fin del documento, partiendo de la posiciôn actual. (Incluso posicionarlo al inicio o fin del documento).

- Reducir/Agrandar el tamaño de la representaciôn interna. (Siempre se agranda o reduce para dar cabida o eliminar un renglôn, segûn el caso).

- Mover el apuntador de fin un nûmero determinado de bytes. (Con el fin de agrandar o reducir el tamaño del documento).

- Mover el contenido de la representaciôn interna para que pueda ser compilada.

- Hacer que my apunte al primer carâcter del renglôn siguiente/anterior.

- indicar en el status si my se encuentra en el inicio o fin del documento.

- Transferir el contenido del renglôn (o nûmero de renglones) al que apunta my al almacên de exhibiciôn.

4.1.3 Procesador de instrucciones

El procesador de instrucciones es una rutina cuya funciôn principal es ejecutar continuamente un ciclo, en el cual se analiza lo que el usuario tecleô. Si lo que se tecleô fuê un instrucciôn, y êsta existe, se ejecuta; si no existe no hace nada. Si lo que se tecleô fuê un carâcter alfanumêrico dicho carâcter se incluye en el texto en la forma en que lo indique el status (escribiendo encima de lo anterior, que pueden ser espacios si es que el renglôn es nuevo, o insertando el carâcter entre los que se encuentra).

La rutina que procesa instrucciones en un lenguaje tipo Pascal se verîa asi:

```
PROCEDURE  editor
 actualiza contenido de pantalla

 DO
    lee-tecla
    IF     tecla = carâcter de control
      THEN
          busca la instrucciôn en el diccionario
          IF     instruccion existe
            THEN
               ejecutala
            NOELSE
          ENDIF
      ELSE
         texto ( tecla )     % insertar carâcter en el texto
         bandera de hubo cambio en texto
   ENDIF

 FOR EVER

END PROCEDURE
```

La rutina "texto" se encarga de modificar el contenido de el almacên de ediciôn, actualizar mx y actualizar el renglôn, que se estâ modifi - cando, en la pantalla.

4.1.4 Dispositivo de salida

El grado de independencia que se tiene con respecto al dispositivo de salida es muy alto. Si êste cambia no es necesario reprogramar todo el editor, basta con modificar las rutinas para manejo de pantalla de acuerdo al tipo de dispositivo que se tenga.

En el caso de que se tenga una pantalla de refrescamiento, la estructura de datos a utilizar para implantar el almacên de exhibiciôn serâ del siguiente tipo:

Memoria de la pantalla		lo que se despliega en la pantalla
inicio del ârea de refrescamiento.	<-	Bytes que no se ven en la pantalla.
1er byte 1er renglôn	<-	1er byte visible del 1er renglôn.
79 bytes (aprox.)	<-	# bytes visibles horizontalmente (define la longitud en bytes del renglôn).
x bytes	<-	# bytes invisibles horizontalmente (define el nûmero de bytes que se debe evitar antes de iniciar el renglôn siguiente).
1er byte 2o renglôn	<-	1er byte visible del 2o renglôn.
. . .		
1er byte ultimo renglon (24 aprox.)	<-	1er byte visible del ûltimo renglôn.
x bytes		
ûltimo byte del ûltimo renglôn.	<-	ûltimo byte visible del ûltimo reng.
r bytes que pertenecen todavia a el ârea de refrescamiento.	<-	bytes que son invisibles.

Fig. 4.3 Estructura del almacên de exhibiciôn

Las rutinas que manejan êsta estructura, y por lo tanto dependen del del dispositivo, permiten:

- Poner o leer el cursor en cualquier punto de la memoria, que tenga una representaciôn dentro de la pantalla, en coordenas x, y.

- Insertar renglônes. tomando como referencia la posiciôn actual del cursor. (Se da la impresiôn de que se habre un renglôn entre la lînea en la que se encuentra el cursor y la siguiente).

- Borrar el renglôn en el que se encuentra actualmente el cursor. (Se da la impresiôn en la pantalla de que el renglôn desaparece)

- Dejar en blanco un renglôn.

- Recorrer todo el contenido de memoria hacia un lado u otro. (Se da la impresiôn en la pantalla de que se recorre todo el documento)

- Borrar el contenido de la memoria. (Borrar la pantalla)

- Actualizar completamente el contenido de la memoria. (Se muestra informaciôn diferente en toda la pantalla)

- Poner el letrero de fin, siempre que sea necesario, despuês del ûltimo renglôn del documento.

4.1.5 Dispositivo de entrada

El grado de independencia respecto del dispositivo de entrada es muy alto ya que solo se usa una rutina que se encarga de leer la tecla que presionô el usuario.

El movimiento a travês del documento se realiza usando las flechas que tiene el teclado. Tambiên se puede usar un bâston (Joy-stick) para êsto ûltimo.

4.1.6 Manejo de archivos.

Las rutinas que hacen el manejo de archivos dependen en gran medida de la estructura del sistema operativo y el acceso que se tenga a el directorio del disco.

Lo que estas rutinas permiten hacer consiste bâsicamente:

- Localizaciôn del nombre de un archivo en el directorio.

- Leer el contenido de un archivo (leer todos los sectores que contengan el archivo)

- Escribir el contenido de un archivo (grabar el contenido de memoria en los sectores del disco y modificar el directorio para que se actualize el nûmero de sectores y la direcciôn de cada uno, bajo el nombre del archivo)

- Crear un archivo con el nombre dado por el usuario.

La informaciôn se graba en disco es una copia idêntica de la representaciôn interna del documento.

4.2 Descripciôn de las rutinas del editor.

En êsta parte se describen las instrucciones con las que se implantô el editor de pantalla.

En la descripciôn se usa la notaciôn siguiente:

a) Funciones

<compilador usado en su creacion> nombre <parâmetros> ;

Descripciôn de la funciôn.

b) Variables

<valor inicial> <compilador usado en su creaciôn> nombre

Uso que se le da.

c) hex

Indica base 16 para los nûmeros usados a partir de donde encuentra.

d) decimal

Indica base 10 a partir de donde se encuentra.

4.2.1 Funciônes principales.

:: edita ;

Inicia una sesiôn de ediciôn. Si el nombre que se da se encuentra en el directorio, lo accesa. Si no crea un documento nuevo con ese nombre. Inicializa la representaciôn interna, el renglôn, llama a la rutina 1editor.

:: 1editor ** car 1dp ;

Lee la tecla que el usuario presionô, si es de control entonces ejecuta la instrucciôn correspondiente. En caso contrario inserta el texto en el renglôn.

0 global inicio

m: inicio inicializa ;

Habilita el modo de inicializaciôn.

m: inicio noinicio ;

Deshabilita el modo de inicializaciôn.

0 global memoria

Variable a la que se asocia el documento.

hex

fbff global fmem

Valor del lîmite superior de la memoria.

m: memoria imem ;

Inicializa todas las variables necesarias para la ediciôn de un documento.

4.2.2 Funciones que modifican o se ejecutan en la representaciôn interna.

ffff global tope

Valor inicial del apuntador que apunta al tope del archivo.
Se le da êste valor inicial para evitar conflictos con dp y el espacio que se le da a la representaciôn interna.

0 global entope

Su contenido indîca si my esta en la primera lînea del archivo.

0 global fin

Apuntador al fin del documento.

0 global my

Apuntador a las lîneas del documento.

:: #cr >> inicio final << num ;

Cuenta el nûmero de retornos de carro entre las direcciones a las que apuntan inicio y final. (Si inicio y/o final apuntan a un separador estos NO se incluyen en la cuenta)

:: fill >> dir numb byte ;

LLena con un carâcter dado en byte tantos bytes como numb empezando en dir.

:: en >> valor ini fin << p ;

Pregunta si esta dentro del rango [ini, fin] .
Regresa un 1 si esta en el rango, 0 en el caso contrario.

:: }cr >< dir ;

Busca un ĉr^, en la direcciôn tope -> fin .
Si lo encuentra deja en la pila la direcciôn en la que se encuentra.
En caso contrario imprime el letrero " ERROR EN cr " y aborta el programa.

:: {cr >< dir ;

Busca un ĉr^, en la direcciôn tope <- fin .
Si lo encuentra deja en la pila la direcciôn en la que se encuentra.
En caso contrario imprime el letrero " ERROR EN cr " y aborta el programa.

m:: tope itope >> dir ;

Inicializa el apuntador tope. tope apuntarâ a la direcciôn que dê como resultado la suma: la direcciôn a la que apunta dp + 101h. El contenido de êsa direcciôn serâ un cr.

m: my imy ;

Inicializa el apuntador my. my apuntarâ a la direcciôn que dê como resultado la suma: la direcciôn a la que apunta tope + 1.

m: tope !tope ;

Le asigna el valor que esta en la pila a tope.

m:: fin ifin >> dir ;

Inicializa el apuntador fin. fin apuntarâ a la direcciôn que dê como resultado la suma: la direcciôn a la que apunta tope + 1. El contenido de esa direcciôn serâ un cr.

m: fin !fin ;

Le asigna el valor que esta en la pila a fin.

m: fin +fin ;

Mueve el apuntador de fin el nûmero de bytes que este en la pila, en la direcciôn tope -> fin.
Si el resultado de la suma de la direcciôn a la que apunta y 200h es mayor que la direcciôn a la que apunta fmem imprime el letrero .: "memoria casi llena".

En general se usa cuando se va a insertar un renglôn en la representaciôn interna y entonces se le pasa como parâmetro la longitud del renglôn nuevo.

m: fin -fin ;

Mueve el apuntador de fin el nûmero de bytes que esten en la pila, en la direcciôn tope <- fin.
En general se usa cuando se borra un renglôn en la representaciôn interna y entonces se le pasa como parâmetro la longitud del renglôn que se va a borrar.

m: my decmy ;

Decrementa en uno el apuntador my.

m: my incmy ;

Incrementa en uno el apuntador my.

: qfuente ;

Mueve la representaciôn interna a la parte alta de la memoria. Para que pueda ser compilada.

: tfuente ;

Posiciona la representaciôn interna del documento en el lugar que ocupaba originalmente, para reiniciar la ediciôn.

:: achicar >> numbytes ;

Reduce el tamaño de la representaciôn interna. Copia el bloque (*) de memoria que inicia en [my + numbytes] y termina en [fin] a partir de my. (Con lo cual ahora el apuntador fin estarâ apuntando a [fin - numbytes]). En general se usa para borrar lîneas de la represen - taciôn interna.

* (Se hace uso de la instrucciôn: ldir de Z80 implantada en Forth)

:: agrandar >> numbytes ;

Agranda el tamaño de la representación interna. Copia el bloque (*) de memoria que inicia en [fin] y termina en [my] a partir de [fin + numbytes] (Con lo cual ahoral apuntador fin estará apuntando a [fin + numbytes]). En general se usa para insertar líneas en la representación interna.

* (Se hace uso de la instrucción: lddr de Z80 implantada en Forth)

m:: my -my >> dir ;

Salva el contenido de la línea actual y hace que my apunte al primer carácter de la línea anterior. Transfiere el contenido de dicha línea, al renglón (almacén de edición).

m:: my >my >> dir ;

Hace que my apunte al primer carácter de la línea siguiente.

m:: my <my >> dir ;

Hace que my apunte al primer carácter de la línea anterior.

m:: my +my >> dir ;

Salva el contenido de la línea actual y hace que my apunte al primer carácter de la línea siguiente. Transfiere el contenido de dicha línea, al renglón (almacén de edición).

: }my ;

Hace lo mismo que +my y además actualiza la posición del cursor.

: {my ;

Hace lo mismo que -my y además actualiza la posición del cursor.

m:: my myasig >> car dir ;

Se inserta el carácter que esta en la pila, en la representación interna en el lugar al que apunta my. Generalmente se usa para insertar una línea entre el texto. Para lo cual basta con insertar un cr, ya que éste carácter marca el inicio o final de una línea.

m: my .my ;

Le asigna a my el valor de la dirección, que se encuentra en la pila. Salva el contenido del renglón que se estaba editando y una vez que tiene su nueva posición transfiere la nueva línea a el renglón.

m: entope sientope ;

Ésta bandera índica que my apunta la primera línea del archivo.

m: entope noentope ;

Ésta bandera índica que my no apunta la primera línea del archivo.

4.2.3 Funciones que modifican o se ejecutan en el renglôn (almacên de ediciôn).

decimal

100 constant longren

Define la longitud del renglôn.

0 global mx

Apuntador al renglôn.

0 global cambio

Indica que hubo un cambio en el renglôn, con respecto al contenido anterior.

0 global vdist

Almacena la distancia del renglôn anterior, generalmente del que se esta re-editando.

: renglôntp ;

Le asigna a renglôn 100 bytes en el diccionario .Un byte antes del inicio del renglôn se encuentra el tamaño del renglôn. Esta dado por #columnas.

m: mx !mx ;

Le asigna el valor que esta en la pila a mx.

m: mx imx ;

Inicializa el apuntador mx, es decir, mx apuntarâ al inicio del renglôn. (la posiciôn del cursor tambiên la pone al inicio del renglôn de la pantalla).

m:: mx .mx >> val dir ;

Se modifica la posiciôn de mx, su nueva posiciôn esta dada por val. Al mismo tiempo actualiza la posiciôn del cursor.

m: mx incmx ;

Se incrementa en uno la posiciôn de mx.

m:: mx mxasig >> val dir ;

Asigna un carâcter al renglôn en la posiciôn mx, avanza mx y cx .

m: renglôn distancia

Para obtener la posiciôn actual del apuntador al renglôn.

m: cambio sicambio ;

Se usa para indicar que el renglôn actual fuê modificado.

m: cambio nocambio

Se usa para indicar que el renglôn actual no fuê modificado.

m: renglôn fren ;

Da como resultado la posiciôn en la que se encuentra el ûltimo carâcter , en el renglôn que se estâ editando.

m: vdist calcula ;

Variable que almacena la longitud inicial del renglôn que se va a editar.

m: vdist incvdist ;

Incrementa el valor de vdist en uno.

m: vdist Ovdist ;

Le asigna a vdist el valor de cero.

m: renglôn lren ;

Limpia el renglôn conservando su distancia anterior en vdist.

m: renglôn iren ;

Inicializal renglôn, dejandolo en blanco.

:: agrandaralmacên >> numbytes ** dir ;

Agranda el tamaño del renglôn. Copia el bloque de memoria que inicia en [fren] y termina en [mx] a partir de [fren + numbytes] (*). En general se usa para insertar caracteres en el renglôn.

*(Se hace uso de la instrucciôn: lddr)

:: achicaralmacên >> numbytes ** dir ;

Reduce el tamaño del renglôn. Copia el bloque (*) de memoria que inicia en [mx + numbytes] y termina en fren a partir de mx. (Con lo cual ahora el apuntador fren estarâ apuntando a [fren - numbytes]). En general se usa para borrar caracteres del renglôn.

* (Se hace uso de la instrucciôn: ldir)

O global imodo

El contenido de esta variable îndica si esta habilitado o deshabilitado el modo inserciôn de caracteres.

m:: renglôn insercion >> carâcter posiciôn ;

Inserta un carâcter en la posiciôn a la que apunta mx. Avanza mx y la posiciôn del cursor: cx. Refresca el renglôn, en el almacên de exhi - biciôn para que se note el cambio.

m: imodo sinserta

Habilita el modo inserciôn de caracteres.

m: imodo noinserta

Deshabilita el modo inserciôn de caracteres.

:: texto >> carâcter ;

Almacena en renglôn el carâcter que estâ en la pila.
Si esta en modo inserciôn usa la rutina de inserciôn de caracteres.
En caso contrario asignal carâcter a la posiciôn del renglôn a la que apunta mx, hace eco en la pantalla, avanza mx y cx.

:: insertacadena ** carâcter ;

Inserta el contenido del almacên que usa inline a partir de la posiciôn de mx.

:: reemplazacadena ** caracter aplbp ;

Supone que mx apunta al inicio de la cadena que se quiere reemplazar.
Borra la cadena a la que apunta mx, usando la cadena a la que apunta lbp como guia, despuês inserta la cadena a la que queda apuntando lbp.
Se tiene el siguiente formato en el almacên:
Cadena a ser reemplazada / cadena que reemplaza

:: buscayreemplazacadenas ** carâcter ap aplbp poscadena ;

Busca y reemplaza todas la instancias de la cadena a ser reemplazada por la cadena que reemplaza en el renglôn.

:: boinrenglôn >> poscadena ** car ;

Transfiere el contenido de la lînea a la que apunta my al renglôn, reemplaza la primera instancia de la cadena en el renglôn y despuês llama a la rutina buscayreemplazacadenas.

4.2.4 Funciones que permiten la transferencia de datos entre el renglôn y la representaciôn interna.

:: 1trans >> longitud ;

Transfiere el renglôn que se esta editando hacia la representaciôn interna. A partir del lugar al que apunta my.
El parâmetro de entrada indica la longitud del renglôn.

:: 2trans ;

Transfiere la lînea, a la que apunta my, al renglôn para poderla modificar. Pone en blanco el renglôn antes de hacer la transferencia.

:: }trans ;

Transfiere la lînea, a la que apunta my, al renglôn para poderla modificar. Pone en blanco el renglôn antes de hacer la transferencia.

Pone en vdist la longitud del renglôn y pone la bandera de cambio en cero (no cambio).

:: {trans ** nlong dlong ;

Transfiere el contenido del renglôn a la memoria, si y solo si hubo algûn cambio.

4.2.5 Funciones que muestran el contenido de la representaciôn interna en la pantalla.

4.2.5.1 Funciones independientes del dispositivo de salida.

0 global cx

Indica la posiciôn del cursor en el eje horizontal.

0 global cy

Indica la posiciôn del cursor en el eje vertical.

0 global lfin

Indica el nûmero de renglônes desde la primera lînea de la pantalla hasta el fin del archivo.

```
: casa ;
```

Posiciona el cursor en el extremo superior izquierdo de la pantalla e inicializa el renglôn.

```
m: cx !cx  ;
```

El contenido de la pila es la posiciôn que se le asigna, en el eje horizontal, al cursor.

```
m:: cx .cx >> val dir ;
```

Se modifica la posiciôn del cursor, su nueva posiciôn esta dada por val. Al mismo tiempo actualiza la posiciôn de mx.

```
m: cx Ocx ;
```

Posiciôn del cursor en el primer carâcter de la lînea.

```
: >movimiento ;
```

Avanza una posiciôn, a la derecha en la pantalla, el cursor.

```
: <movimiento ;
```

Retrocede una posiciôn, a la izquierda en la pantalla, el cursor.

```
m: cy Ocy ;
```

Posiciôn del cursor en el primer renglôn de la pantalla.

```
m: cy .cy  ;
```

Le asigna a la posiciôn del cursor, en el eje vertical, el valor que haya en la pila.

```
:: }pantalla   ** dir ;
```

Copia el contenido del renglôn en la pantalla.

```
m: lfin ilfin ;
```

Inicializa la posiciôn del letrero de fin (poniendolo en el segundo renglôn de la pantalla).

```
m: lfin .lfin  ;
```

Le asigna a la posiciôn del letrero de fin el valor de lo que hay en la pila.

```
m:: lfin -lfin >> dir ;
```

Decrementa en uno la posiciôn del letrero de fin en la pantalla.

```
m:: lfin +lfin >> dir ;
```

Incrementa en uno la posiciôn del letrero de fin en la pantalla.

```
:: implfin >> y ;
```

Imprime el letrero de fin en la pantalla en el renglôn que îndica el contenido en la pila.

```
:: ?fin << sifin ;
```

Regresa un 1 siempre que se encuentre en la ûltima lînea del archivo. 0 en el caso contrario.

```
m:: cy -cy >> dir ;
```

Se usa para viajar en el documento en la direcciôn (fin -> tope).

Hace las siguientes consideraciones:

Si esta en el tope del archivo ahî se detiene.

Si esta en el renglôn cero de la pantalla y aûn hay documento por ver, hace "scroll" hacia abajo.

Cuida de la impresiôn del letrero de fin cuando es necesario.

```
m:: cy >cy >> dir ;
```

Se usa para viajar en el documento en la direcciôn (tope -> fin).

Hace las siguientes consideraciones:

Si esta en el ûltimo renglôn de la pantalla ahî se detiene.

Si esta en el ûltimo renglôn de la pantalla y aûn hay documento por ver, hace "scroll" hacia arriba.

Cuida de la impresiôn del letrero de fin cuando es necesario.

```
m:: cy +cy >> dir ;
```

LLama a >cy siempre y cuando no estê en la ûltima lînea del archivo, al mismo tiempo que esta en la ûltima lînea de la pantalla.

```
:: refrescar ** contador apuntador >> dii ;
```

Transfiere a la pantalla, a partir de dii, que generalmente es la direcciôn del primer carâcter de una lînea, todas las lîneas que sean necesarias para llenarla.Si se estâ al final del archivo imprime el letrero de fin donde corresponde.

```
:: rlinea >> dirmy ** dlfin dcy ;
```

Si el cursor estâ en la ûltima lînea del documento y en el ûltimo renglôn de la pantalla entonces se imprime el letrero de fin, sobre la lînea en la que estal cursor. Si no esta al final del documento pero el cursor estâ en la ûltima lînea de la pantalla transfiere el contenido de la lînea a la que apunta my al renglôn y de ahî a la pantalla. En caso contrario, como se usa en el borrado de lîneas esta funciôn, hay que mandar a la ûltima lînea de la pantalla la lînea correspondiente.

```
: 1bren ;
```

Borra la lînea en la que se encuentra el cursor actualmente. Hace lo mismo con la lînea a la que apunta my. LLama a la funciôn rlinea.

4.2.5.2 Funciones dependientes del dispositivo de salida.

```
decimal
79 global #columnas
24 global #renglônes
:: .xcursor >> x ** y ;
```

Posiciona el cursor en el eje horizontal de la pantalla de acuerdo al valor de x, conservando la posiciôn en el eje vertical.

```
:: .ycursor >> y ** x ;
```

Posiciona el cursor en el eje vertical de la pantalla de acuerdo al valor de y, conservando la posiciôn en el eje horizontal.

```
:: .xycursor >> x y ;
```

Posiciona el cursor, usando como coordenadas los valores x, y.

```
: home ;
```

Posiciona el cursor en el extremo superior izquierdo de la pantalla.

: >echo ;

Mueve una posiciôn a la derecha, en la pantalla, el cursor.

: <echo ;

Mueve una posiciôn a la izquierda, en la pantalla, el cursor.

: vecho ;

Mueve hacia abajo en la pantalla , un renglôn, el cursor. Conservando su posiciôn en x.

: ^echo ;

Mueve hacia arriba en la pantalla , un renglôn, el cursor. Conservando su posiciôn en x.

: insertalinea

Inserta una lînea en la pantalla.Tomando como referencia la posiciôn del cursor.

: borraren ;

Elimina la lînea en la que se encuentra actualmente el cursor.

: borralinea ;

Borra la lînea en la que se encuentra actualmente el cursor, poniendo blancos.

: ^scroll ;

Hace "scroll" hacia arriba. Supone que el cursor està en el ûltimo renglôn de la pantalla y que despuês se actualiza su posiciôn en x.

: vscroll ;

Hace "scroll" hacia abajo. Supone que el cursor està en el primer renglôn de la pantalla y que despuês se actualiza su posiciôn en x.

: borrapantalla ;

Borra la pantalla, al escribir en el almacên de exhibiciôn 20 ASCII.

5. Conclusiones

5.1 Resultados

El objetivo principal del editor de textos se alcanzô ya que permite la creaciôn, modificaciôn y mantenimiento de programas y textos.

La interfaz con el usuario, por las comentarios que se han obtenido de diferentes usuarios es que êsta es suficientemente clara y concisa.

Permite realizar la bûsqueda con o sin reemplazo de un patrôn dado.

Tiene la capacidad de mezclar diferentes documento tales como: grâficas y texto, con facilidad.

El editor realmente forma parte de un sistema integrado, ya que permite al usuario a la mitad de una sesiôn de ediciôn, el obtener informaciôn mirando a travês del sistema de archivos y usar todos los recursos de la mâquina, siempre que respeten elârea de memoria en que se encuentra el archivo, como p.ej. crear rutinas, hacer câlculos, etc. con regreso transparente a la sesiôn de ediciôn.

La creaciôn de rutinas, que modifiquen el documento, fuera de sesiôn permite al usuario desarrollar y depurar las rutinas que fâciliten la ediciôn del tipo de documentos que necesita.

La mâquina en la que se implantô el sistema es una microcomputadora de 8 bits basada en el microprocesador Z80, cuya versiôn mâs simple tiene 64 Kbytes de RAM dinâmica, 8 Kbytes de PROM en donde reside el intêrprete de Forth, 12 Kbytes de RAM estâtica de los cuales 4 Kbytes son para el despliegue de caracteres (resoluciôn de 80 por por 24) y 8 Kbytes para el despligue de grâficas (resoluciôn de 512 por 256), posee un controlador para manejador de discos flexibles de 8 pulgadas, asî como de 5 1/4 pulgadas, con capacidad de 300 Kbytes por disco y 2 puertos paralelos libres para el manejo de equipo perifêrico, p.ej. impresora.

El espacio que ocupa el editor es 8 Kbytes de memoria dejando para los programas de usuario 48 Kbytes de memoria.

Las aplicaciones que ha tenido el editor son diversas, entre las mâs importantes estân:

La creaciôn de utilerîas y textos para la implantaciôn de cursos de diferentes materias, matemâticas y quîmica por ejemplo , los cuales utilizan como herramienta fundamental para la enseñanza la computadora. (Estos cursos estân actualmente siendo impartidos en el Instituto Tecnolôgico de Sonora.)

Implantaciôn de utilerîas como un ensamblador para Z80. Y en base a la pareja editor - ensamblador el desarrollo de manejadores para equipo perifêrico para almacenamiento masivo implantando una interfaz SASI.

(Por medio de esta aplicaciôn se tiene acceso a capacidades de memoria secundaria de decenas de Megabytes en medios magnêticos.)

5.2 Extensiones a Forth

Las extensiones que se le hicieron a Forth probaron ser de utilidad en la implantaciôn de este trabajo.

5.3 Trabajo Futuro

5.3.1 Transportarlo

Se piensa que transportar el editor a un sistema operativo de tipo comercial podrîa ser ventajoso en funciôn de dos de sus caracterîsticas mâs importantes: râpidez de respuesta y la extensibilidad.

5.3.2 Mejoras

Entre la gran cantidad de mejoras que puede tener el editor, serîa deseable que se le agregarâ: la duplicaciôn de parrâfos, la repeticiôn de la instrucciôn dada anteriormente, asi como cortar una linea en dos y unir una lînea con otra, poder insertar archivos en el documento que se esta editando, salvar parte del documento que se esta editando como un archivo.

Generar una versiôn que permita el formateo de textos.

[Bel 73]
Bell J. R.
Threaded code,
Communications, ACM, Vol 16, N 6, Junio 1973, pp 370-372

[BYT 80]
Byte, Vol 5, N 8, Agosto 1980
(Nûmero dedicado a Forth)

[Dew 75]
Dewar R. B. K.
Indirect threaded code,
Communications, ACM, Vol 18, N 6, Junio 1975, pp 330-331

[Dun 81]
Duncan R.
Forth decompiler,
Dr. Dobb^s Journal, N 59, Septiembre 1981, pp 49-53

[Ger 84a]
Guerra O. V.
Introducciôn al Forth,
Comunicaciones Têcnicas serie azul, I.I.M.A.S. - U.N.A.M, 1984

[Ger 84b]
Guerra O. V.
Forth Manual de programaciôn,
Comunicaciones Têcnicas serie azul, I.I.M.A.S. - U.N.A.M, 1984

[Har 80]
Harris K.
Forth extensibility,
Byte, Vol 5, N 8, Agosto 1980, pp 164-184

[Har 81]
Harris K.
The Forth philosophy,
Dr. Dobb^s Journal, N 59, September 1981, pp 6-11

[Kog 82]
Kooge P. M.
An architectural trail to threaded-code systems,
Computer, Vol 15, N 3, Marzo 1982, pp 22-32

[Loe 81]
Loelinger R. G.
Threaded interpretive languages,
Byte books, 1981

[Mey 82]
Meyrowitz N. y Van Dam A.
Interactive Editing Systems,
Computing Surveys, ACM, Vol 14, N 3, Septiembre 1982, pp 321-415

[Ros 82]
Rosenblueth L. D.
Un editor de pantalla implantado sobre un editor de lînea,
Comunicaciones Têcnicas serie naranja, I.I.M.A.S.-U.N.A.M, 1982

[Ros 83]
Rosenblueth L. D.
Un programa auxiliar en el diseño de circuitos: su implantaciôn mediante côdigo hilvanado,
Têsis Maestrîa, U.A.C.P.P del C.C.H., U.N.A.M, 1983

[Ros 84]
Rosenblueth L. D.
Documentaciôn de las extensiones hechas a Forth
I.I.M.A.S.-U.N.A.M, 1984

[Tha 82]
Thanh N. T. y Raschner E. W.,
Indirect threaded code used to emulate a virtual machine,
SIGPLAN Notices, ACM, Vol 17, N 5, Mayo 1982, pp 80-89

[Win 81]
Winston P. H. y Horn B. K. P.
Lisp,
Addison-Wesley, 1981

[Zil 79]
Z-80 Assembly language programming manual,
Zilog, California, 1979

Apêndice

INSTRUCCIONES PARA EL USO DEL EDITOR

1. Para usar por primera vez el editor hay que cargarlo de la siguiente manera:

   Dar: CARGA
   Responder: ARCHIVO: EDITORCP.O

2. Una vez que el el editor se cargo al sistema con la instruccion EDITA se pasa al vocabulario que permite su uso.

   Dar: EDITA
   Responder: ARCHIVO: nombre.S

   Es importante que el nombre del archivo a editar termine con ".S" para indicar que se trata de un archivo "fuente", en contraposicion a los archivos de programas ya compilados que tienen la terminacion ".O" .

3. Usar las instrucciones del editor para crear o modificar el archivo especificado:

   ^T Cursor al tope del archivo.

   ^J Cursor hacia abajo.

   ^K Cursor hacia arriba.

   ^H Cursor hacia la derecha.

   ^L Cursor hacia la izquierda.

   ^Y Cursor al final de la linea.

   ^A Cursor al principio de la linea.

   ^B Borra el caracter indicado por el cursor.

   ^X Borra a partir del cursor el resto de la linea.

   ^Z Borra toda la linea.

   ^V Inicio del modo de insercion.

   ^N Pagina siguiente.

   ^U Pagina anterior.

   ^I Buscar a partir del la posicion del cursor una cadena.

      - dar la cadena -

^O Buscar y reemplazar a partir de la posicion del cursor una cadena.

- cadena:vieja/cadena:nueva -

^Q Terminar la edicion.

^E Regresar al modo editor.

DEL Restaura el contenido del renglon.

4. Despues de salir del editor se debe de guardar en disco el archivo:

```
Dar:        GUARDA
Responer:   ARCHIVO: Nombre.S        (este nombre puede diferir del
                                      especificado en el paso 2).
```

5. Si el archivo corresponde a un programa en forth entonces hay que compilarlo:

```
Dar:        LOAD
```

En caso de detectarse errores hay que corregirlos regresando al editor con la instruccion ^E. Note que esto solo se puede hacer despues de haber cargado el editor.

6. Si se desea almacenar en disco el programa ya compilado entonces es conveniente primero quitar el editor y despues volver a compilar. Asi solo el programa se almaceara.

```
Dar:        FORGET EDITOR
Dar:        LOAD
Dar:        GUARDA:
Dar:        NOMBRE: Nombre.O          (el nombre deve de terminar con O )
                                    ( para que se guarde el programa Objeto )
```

T U T O R P A R A E D I C I Ô N

Funciones	Explicaciôn
: edita	Inicia una sesiôn de ediciôn.
: ^l	Mueve el cursor hacia la derecha sobre el renglôn.
: ^h	Mueve el cursor hacia la izquierda sobre el renglôn.
: ^j	Mueve el cursor en la direcciôn (tope -> fin), pasando de un renglôn a otro.
: ^k	Mueve el cursor en la direcciôn (fin -> tope), pasando de un renglôn a otro.
: ^y	Posiciona el cursor despuês del ûltimo carâcter en el renglôn.
: ^t	Posiciona el cursor en la primera lînea del documento, desplegando a partir de ahî el contenido del documento sobre la pantalla.
: ^m	Êsta funciôn define el retorno de carro.
: ^i "cadena a bûscar"	Êsta funciôn bûsca una cadena en el archivo, a partir de la posiciôn actual de my. Si no la encuentra despliega el fin del documento. El formato de la instruccion es el siguiente: "cadena a bûscar<cr>".
: ^b	Êsta funciôn borra el carâcter sobre el cual se encuentra el cursor.
: ^v	Êsta funciôn permite la insercion de caracteres en el renglôn, a partir de la posiciôn actual del cursor.
: ^p	Êsta funciôn transfiere el contenido del renglôn a la pantalla.
: ^n	Êsta funciôn avanza el cursor la pâgina siguiente.
: ^u	Êsta funciôn posiciona el cursor en la pâgina anterior.

: ^d	Êsta funciôn transfiere el contenido del renglôn hacia la representacion interna, y le pasa el control a FORTH. Generalmente se usa para depuraciôn de rutinas .
: rubout [del]	Transfiere el contenido de la lînea a la que apunta my, hacia el renglôn y de êste hacia la pantalla, con lo cual se olvidan todos los cambios que se le hayan hecho al texto hasta ese momento.
: ^a	Manda el cursor al inicio del renglôn.
: ^f	Manda el cursor al final del texto que hay en el renglon
: ^z	Êsta funciôn borra la lînea en la que se encuentra actualmente el cursor.
: ^o	Êsta funciôn permite hacer bûsqueda y reemplazo de todas las instancias de una cadena por otra. El formato de la instrucciôn es el siguiente: "cadena a ser reemplazada/cadena reemplazante<cr>"
: ^q	Êsta funciôn posiciona el documento en la parte alta de la memoria para ser compilado. El inicio de la compilacion esta dado por TOPE.
: ^e	Êsta funciôn regresa el documento a su posiciôn original dentro de memoria despuês de haber sido compilado, para continuar editandolo.

Ejemplo, suponga que usted tiene el texto siguiente:

"El cohete

Una vara de carrizo delgado lleva un canuto de carrizo mâs grueso en la punta, liado con ixtle bien empapado con cola espesa de carpintero. Eso es un cohete. Lo demâs lo hace la pôlvora. Para los de luces hay que conocer muy bien los secretos del oficio, como don Atilano. A la pôlvora se le agregan sales metâlicas, de cobre, de hierro, de aluminio, segûn el color que se quiera. Hacer un castillo es ya otra cosa. Hay que tener muchos conocimientos y buenas ocurrencias de arte mecânico. Sobre todo para un castillo como el que van a quemar el dîa de la funciôn, que serâ mâs alto que la parroquia. Eso es ya cosa de arquitectura.

Yo vi el dibujo. Cuatro torres sostienen una plataforma a ocho metros del suelo.

Desde allî se alza el castillo propiamente dicho, como el tronco del pino mâs alto que haya en toda la sierra. Va a dar vuelta todo entero, movido por unas aspas de luz amarilla y verde, los colores del señor San Josê, y en la mera punta se descubrirâ, al final, una imâgen de nuestro santo patrono, sobre una catarata de luz, rodeada por canastillas que saldrân de todas partes, en forma de querubines ...

Se revestirâ siete veces, y don Atilano tiene calculado que llevarâ mâs de quinientas girândulas. Para que la gente no se acerque mucho y vaya a haber un accidente, alrededor de todo el castillo andarân los toritos de fuego que asustan a el pueblo con miles de buscapiês.

Al cabo se podrâ ver desde muy lejos.

Juan Josê Arreola"

Si el cursor esta en la primera lînea del texto:

"Una vara de carrizo delgado lleva un canuto de carrizo mâs grueso"

y presiona las teclas "ctrl" y "F" simultaneamente (^F), el cursor se posicionarâ inmediatamente despuês de la palabra "grueso".

Si ahora se presionan "ctrl" y "A" (^A), el cursor se posicionarâ varios espacios antes de la palabra "Una", al inicio del renglôn.

Para viajar por el texto usted puede usar la flechitas.

Si se quisiera sustituir "Una vara de carrizo delgado" por "Una rama delgada, limpia de hojas y lisa" bastarâ que teclee ^T para situarse en el tope del documento. Y despuês ^O con lo cual se borrarâ la ultima lînea de la pantalla y el cursor se posicionarâ ahî. Hecho esto se deberâ teclear lo siguiente:

"Una vara de carrizo delgado/Una rama delgada, limpia de hojas y lisa<cr>
(cadena a ser reemplazada)/(cadena reemplazante)<cr>"

Con lo cual el texto quedarâ :

"Una rama delgada, limpia de hojas y lisa lleva un canuto de carrizo mâs grueso".

Suponiendo que usted viaja a travês del texto y posiciona el cursor sobre la lînea que acabamos de modificar.

Si ahora quiere usted cambiar por ejemplo las palabras "limpia de" por la palabra "sin" lo puede hacer de la manera siguiente :

Posicione el cursor en el inicio de la palabra "limpia" y presione ^B con ello se iran borrando los caracteres, uno tras otro. Una vez que haya borrado las palabras "limpia de" el texto de la lînea se habra reducido. Ahora usted deberâ presionar ^V para entrar a el modo de insercion y teclear "sin" se irân insertando los caracteres que usted teclea incrementando el tamaño de la lînea. Despuês de haber tecleado "sin" presione ^j para salir del modo inserciôn y pasar al renglôn siguiente.

Con lo cual el texto quedarâ :

"Una rama delgada, sin hojas y lisa lleva un canuto de carrizo mâs grueso"

Supongamos que queremos encontrar la palabra "ixtle" entonces nos vamos a el tope del documento, para que a partir de ahî se haga la bûsqueda, con ^T, despuês se presiona ^I con lo cual se borrara la ultima lînea de la pantalla y el cursor se posicionarâ ahî. Hecho esto usted deberâ teclear lo siguiente:

ixtle<cr>
(cadena buscada)<cr>

Entonces en el primer renglôn de la pantalla aparecera :

"en la punta, liado con ixtle bien empapado con cola espesa de carpintero. Eso es un cohete. Lo demâs lo hace la pôlvora. Para los de luces hay que conocer muy bien los secretos del oficio, como don Atilano. A la pôlvora se le agregan sales metâlicas, de cobre, de hierro, de aluminio, segûn el"

.
.
.

Si quisieramos insertar una lînea entre las primeras dos : el cursor deberâ estar en la que dice: "en la punta, liado con ixtle ... " y presionar tan solo la tecla return ,<cr>, y el texto quedarâ:

"en la punta, liado con ixtle bien empapado con cola espesa de carpintero.

Eso es un cohete. Lo demâs lo hace la pôlvora. Para los de luces hay que conocer muy bien los secretos del oficio, como don Atilano. A la pôlvora se le agregan sales metâlicas, de cobre, de hierro, de aluminio, segûn el"

.
.
.

Si no nos gusta y deseamos borrar dicha lînea, basta posicionar el cursor ahî y teclear ^Z para que esta lînea sea borrada. El texto quedarâ asi:

"en la punta, liado con ixtle bien empapado con cola espesa de carpintero. Eso es un cohete. Lo demâs lo hace la pôlvora. Para los de luces hay que conocer muy bien los secretos del oficio, como don Atilano. A la pôlvora se le agregan sales metâlicas, de cobre, de hierro, de aluminio, segûn el"

.
.
.

Si usted modifica un renglôn y no le gustan las modificaciones que le hizo y aun no ha cambiado de renglôn basta que presione la tecla : DEL o RUBOUT para que el renglôn vuelva a su estado original.
Por ejemplo si el cursor esta en la lînea que dice "Eso es un cohete..." y cambia "cohetôn" por "cohete" tendra:

"Eso es un cohetôn. Lo demâs lo hace la pôlvora. Para los de luces hay que"
^
¦
la flecha indica la posiciôn del cursor

y suponiendo que se quiere volver a la situaciôn original, basta con presionar DEL para tener:

"Eso es un cohete. Lo demâs lo hace la pôlvora. Para los de luces hay que"

Si usted ahora tratara de compilar este texto basta tan solo con que presione ^Q. Con esto pasa a FORTH lo ûnico que tiene que hacer ahora es teclear LOAD para que el texto sea compilado. El resultado de la compilaciôn es fâcil predecirlo, sera:

"El ?"

Ya que esa funciôn no esta definida en Forth.

Para regresar despuês de esto a seguir editando el texto basta tan solo que se presione ^E . Para volver a editar un nuevo documento teclee la instrucciôn "edita" y se iniciarâ una nueva sesiôn.

Listados del programa

```
: editor ;
vocabulary editando
editando definitions
hex
dfff global fmem    % ultima posicion libre en memoria

% ------ manejador de pantalla -------
decimal
25 global #renglones        % numero visible de renglones
19 global tpagina           % taman~o de pagina <- (#renglones 5 - )
80 global #caracteres       % numero de caracteres visibles / renglon
128 global #car/renglon     % numero de caracteres reales / renglon
hex
E000 constant ipantallaR    % primer caracter Real
1000 constant memvideo      % tamano de la memoria de video

E198 constant ipantalla     % primer caracter visible

0 global fpantalla          % ultimo caracter visible ( EF16h )
m: fpantalla !fpantalla ! ;
0 global #columnas
m: #columnas !#columnas ! ;
#caracteres 1- !#columnas
#renglones 1- #car/renglon * ipantalla +
#caracteres + !fpantalla
0 global #bytes              % numero total de bytes visibles
m: #bytes !#bytes ! ;
#renglones #car/renglon * !#bytes
0 global cx
0 global cy
hex
2F constant slash
:: .xcursor >> x ** y &
    reng @ x + cursor !
;
:: .ycursor >> y ** x &
   cursor @ reng @ - .x
   #car/renglon y * ipantalla + reng !
   reng @ x + cursor !
;
:: .xycursor >> x y &
   #car/renglon y * ipantalla + reng !
   reng @ x + cursor !
;
: home
  ipantalla dup reng ! cursor !
;
hex
: ^echo
  reng @ #car/renglon - reng !
  cursor @ #car/renglon - cursor !
;
```

```
: vecho
  reng @ #car/renglon + reng !
  cursor @ #car/renglon + cursor !
;
: borraren
  reng @ reng @ #car/renglon + #car/renglon reng @ fpantalla - - ldir
;
: borralinea
  20 reng @ c!
  reng @ 1+ reng @ #car/renglon 1- ldir
;
: insertalinea
  fpantalla fpantalla #car/renglon - fpantalla reng @ - 1+ lddr
  borralinea
;
hex
: ^scroll
  ipantalla ipantalla #car/renglon + #bytes #car/renglon - ldir
                                % supone que el cursor esta hasta abajo y
                                % que despues se actualiza su posicion en x
;
hex
: vscroll
  fpantalla dup #car/renglon - #bytes #car/renglon - lddr
  ipantalla dup                 % supone que x se actualiza despues
  reng ! cursor !
;
hex
: borrapantalla
  20 ipantallaR C!
  ipantallaR 1+ ipantallaR memvideo ldir
;

% ----------------- fin manejador de pantalla -------------



% ----------------- incio de codigo comun ------------

hex
ffff constant terminador
decimal

: posteriormente cabeza 2 allot literal return , ;  % para definir funciones

: completa ^^2+ ! ;                                 % cuyo codigo se completa
                                                    % posteriormente.
: fi here over - swap ! ; immediate

: w> u> ; : w< u< ;
```

```
: trae                          % origen: manejador de archivos.
flag c@
0 =
if
  15 abort
fi
busca1
if
  0 exttipo
  1 <>
  if
    0
  else
   tsrc trae1 1
  fi
else
  1 exttipo
  1 <>
  if
    0
  else
    abre calcula 1 8 ind c! modo.t agrega 1
  fi
fi ;
: voc <builds entry , does> ;                          % para crear un voc dentro
: ag-voc dup context ! current ! ;                     % del editor.
voc graficacion                 % (editor d texto {editor d graficas} ).
:: #cr >> inicio fin << numero ** longitud caracteres &
   inicio fin =
   if
      -1 .numero
     else
      -1 .numero fin inicio - .longitud
      fin 1+ @ .caracteres
      terminador fin 1+ !                    % terminador
      fin 2+ .fin
      inicio
      <do
        longitud cr
        cpir 0=
        if
          fin over - .longitud
          numero 1+ .numero
         else
          drop caracteres fin 1- ! return
        then
      od>
   fi
;
0 global #cr?                          % numero de renglones en el documento
0 global lfin
m: #cr? +#cr 1+! ;                     % incrementa el numero de renglones
m: #cr? -#cr 1-! ;                     % decrementa el numero de renglones
```

```
m: #cr? !#cr !   ;                    % asigna numero de reglones
m: #cr? i#cr 1set ;                   ( aquí)  % era 1set antes
% indica el numero de renglones desde la primera linea de la pantalla hasta
% el fin del archivo
m: lfin ilfin 1set ;
m: lfin .lfin ! ;
m:: lfin -lfin >> dir &
  dir 1-!
;
m:: lfin +lfin >> dir &
  dir 1+!
;
decimal
% 100 constant longren
256 constant longren

0   constant fdl
hex
:: fill >> dir numb byte &          % llena con un caracter dado en byte
  byte dir c!                       % tantos bytes como numb empezando en dir
  dir 1+ dir  numb 1- ldir
;
:: }cr >< dir  &                          % Busca un ĉr^, hacia arriba.
  dir longren cr cpir
  if [ error en cr ] abort fi
  1- .dir
;
:: {cr >< dir  &
  dir longren cr cpdr                     % Busca un ĉr, hacia abajo
  if [ error en cr ] abort fi
  1+ .dir
;
hex
ffff global tope
m:: tope itope >> dir &
     dp @ 100 1+ + dir !
     cr dir @ c!
;
m: tope !tope ! ;
hex
: intallot ( allot inteligente)
   dp @ 80 + tope w>
   if cret [ MEMORIA LLENA ] cret abort fi
;
^intallot memlle !
0 global fin
hex
m:: fin ifin >> dir &
   tope 1+ dir !
   cr dir @ c!
;
m: fin !fin ! ;
```

```
m: fin +fin  +!
   fin 200 + fmem w>
   if  cret [ MEMORIA CASI LLENA ] cret  fi
;
  ( se le pasa como parametro la long del renglon nuevo )

m: fin -fin -! ;
  (      "                              "          viejo )

: qfuente              % hace espacio para los descriptores
   fmem 7 -            % destino
   fin                 % fuente
   fin tope - 1+       % numero de bytes
   lddr
   fmem 7 - fin - tope + !tope
   fmem 7 - !fin
;
hex
: tfuente
    dp @ 100 + 1+        % destino
    tope                 % fuente
    fin tope - 1+        % numero de bytes
    ldir
    fin tope - %  numero de bytes
    dp @ 101 + !tope
    tope + !fin           % fin = tope + numero de bytes
;
hex
: renglontp
  <builds #columnas 1- c,
  ( caracter distinto de espacio)
  longren allot
  does> 1+
;
% Le asigna a renglon 100 bytes
% en el diccionario.
renglontp renglon

0 global mx                          % Esta variable permite accesar el renglon
                                     % que es la imagen del que se esta editando
m: mx !mx ! ;
m: cx !cx over .xcursor ! ;
m:: cx .cx >> val dir &
    val #columnas 1- > if return fi
    val 0< if return fi
    val !cx
    renglon val + !mx
;
m: mx imx renglon swap ! 0 !cx ;

m: mx ^mx  @ c! ;
```

```
m:: mx .mx >> val dir &
   val dir !
   val renglon - !cx
;
m: mx incmx 1+! ;
m: renglon distancia                    % Para obtener la posicion actual del
   mx - abs                             % apuntador a el renglon.
;
0 global cambio
m: cambio sicambio
   1set
;
m: cambio nocambio
   0set
;
posteriormente rcautomatico             % se completa en la rutina : ^m
posteriormente <trans                   % se completa en larutina  : {trans
m:: mx mxasig >> val dir &              % asigna un caracter a el renglon en la
    val dir @ c!
    distancia 1+ #columnas =
    if
        rcautomatico                    % inserta una linea
      else
        mx 1+ .mx
    fi
;
m: renglon fren % >< dir  &     % no hay que pasarle el parametro
    longren 1- + 20<- 1+
;

0 global vdist
m: vdist incvdist dup @ 1+ swap ! ;  % incrementa el valor de vdist en uno
m: vdist 0vdist 0set ;
m: vdist calcula              % variable que almacena la longitud inicial
   fren renglon - swap !      % del renglon que se va a editar.
;
m: renglon lren               % limpiar el renglon conservando su distancia
   1+ longren aspace fill     % anterior en vdist.
;
m: renglon iren                         % Inicializar el renglon con blancos .
  1+ ( salta el num de car) longren aspace fill
  0vdist
;
:: }pantalla   ** dir & ( copia de renglon -> pantalla)
   renglon   .dir
   0 .xcursor
   renglon 1- type
   cx .xcursor
;
0 global my
m: my decmy 1-! ;       % decrementa el apuntador my .
m: my incmy 1+! ;
```

```
:: 1trans >> longitud  &
    my                                  % destino
    renglon                             % origen
    longitud                            % tamano del bloque
    ldir
;
:: 2trans &
   iren
   renglon                                      % destino
   my                                           % origen
   my }cr my -                                  % numbytes
   ldir
;
:: }trans &
   2trans
   calcula
   nocambio
;
:: achicar >> numbytes &
   numbytes
   if
    my                % destino
    my numbytes +     % bloque origen
    dup fin swap -    1+ % tamano del bloque
    ldir
    numbytes -fin     % fin apunta al primer byte libre
   fi
;
:: agrandar >>  numbytes  &
   numbytes
   if
    fin numbytes +   % destino
    fin              % origen
    fin my - 1+      % tamano del bloque
    lddr
    numbytes +fin
   fi
;
:: agrandarbuffer >> numbytes ** dir &  % Para aumentar el tamano del renglon
    fren 1-  .dir
    numbytes
    if
      mx dir w>
      if return fi
      dir numbytes + renglon longren + w<
      if
        dir  numbytes +  % destino
        dir              % origen
        dir mx -  1+     % tamano del bloque
        lddr
      fi
    fi
;
```

```
decimal

: implfin >> y &
  0 y .xycursor
  [ fin ]
  cx y .xycursor
;
:: {trans ** nlong dlong &      % reng -> memoria
   cambio
   if                                % si hay cambios
      fren renglon - .nlong          % numero de caracteres en el renglon
      nlong vdist - .dlong
      dlong 0<
            if      % si la long es neg hay que achicar
               dlong minus achicar
               nlong 1trans
             else    % la long es mayor que cero hay que agrandar
               dlong agrandar
               nlong 1trans
            fi
   fi                % no hay cambios dejalo intacto
   nocambio          % inicializa la variable
;
completa {trans <trans    % origen en : mxasig

: cy 0cy 0set ;
: cy .cy ! ;

0 global entope

: entope sientope 1set ;
: entope noentope 0set ;
:: cy -cy >> dir &
  my 1- tope = if
    entope if return fi
    sientope
  fi
  cy 0=
  if
    vscroll
    +1fin
    }pantalla
  else
    dir 1-!
    1fin cy 1+ = if cy 1+ borralinea implfin fi  ( aquí)
    ^echo
  fi
;
```

```
:: cy >cy >> dir &
  noentope
  cy #renglones 1- =
  if
     ^scroll
     -1fin
     1fin #renglones 1- =
     if
       -cy
     else
       }pantalla
     fi
  else
     dir 1+! vecho
  fi
;
m:: cy +cy >> dir &
 1fin cy 1+ = if
   cy #renglones 1- <> if
     return
   fi
 fi
 >cy


m::  my -my >> dir &
 my 1- tope = if return fi
    my fin > if else {trans fi
    dir 2-!
    dir @ {cr 1+ dir !
    }trans
  ;
  m:: my >my >> dir &          % my -> sigreng
    my c@ cr =
    if
      incmy
    else
      dir @ }cr 1+ dir !
    fi
  ;
  m:: my <my >> dir &
    my 1- c@ cr =
    if
     else
         dir @ {cr 1+ dir !
    fi
  ;
  :: ?fin << sifin &
    0 .sifin
    1fin cy 1+ = if 1 .sifin fi
  ;
```

```
m:: my +my >> dir &
   {trans
   >my
   my fin > if -my else }trans fi
;
: }my
   +my
   +cy
;
: {my
   -my
   -cy
;
: }cy }my ;
: {cy {my ;
m: my imy tope 1+ swap ! ;
m:: my myasig >> car dir & ( para insertar lineas)
   car dir @ c!   % supone que en s esta el byte a asignar
;

0 global memoria
0 global inicio

m: inicio inicializa 1set ;
m: inicio noinicio 0set  ;
m: memoria imem drop inicializa itope imy imx iren ifin ilfin i#cr ;
0 global liga
:: achicarbuffer >> numbytes ** dir &
   fren .dir
   mx dir >= if return fi
   mx ( destino)
   mx numbytes + ( fuente)
   dup dir swap - ( tamano del bloque)
   ldir
   dir numbytes - ( dir inicial)
   numbytes
   aspace fill
;
m: my .my {trans ! my fin w> if else }trans fi ;
m: cx 0cx 0set ;
: casa 0cy 0cx imx home ;
m:: my myultimo? << resultado >> dir &     % 1 = si 0 = no
    0 .resultado                           % supone que si
    dir @ fin >=
    if
     else
       dir @ }cr 1+ fin <
       if
         1 .resultado                      % no es el ultimo renglon
       fi                                  % del documento
    fi
;
```

```
:: refrescar **  apuntador >> dii &
   dii .apuntador
   borrapantalla
   inicio 1 =
   if
      casa cy 1+ implfin casa sientope dii .my
    else
     casa dii .my
     #renglones 1+ 0
     do
       i> #renglones =
       if
         casa apuntador .my }trans leave
        else
         }pantalla
         cy 1+ .cy vecho
         myultimo?
         if
           dii }cr 1+ dup .dii .my
           }trans    % no es el ultimo renglon
          else
           lfin #renglones ( 1-) < if vecho lfin implfin fi
           casa apuntador .my }trans leave
         fi
       fi
     loop
   fi
;

: >movimiento cx 1+ .cx ;
: <movimiento cx 1- .cx ;

: vmovimiento }cy ;
: ^movimiento {cy ;

 0 global imodo

 m: imodo sinserta
     1set
 ;

 m: imodo noinserta
     0set
 ;

 : ^l   >movimiento  ;
 : ^h   <movimiento  ;

 : ^j   vmovimiento noinserta ;
 : ^k   ^movimiento noinserta ;
```

```
: ^t
  {trans
  tope 1+ fin =
  if
     inicializa
   else
     noinicio
     #cr? .lfin
 fi
  tope 1+ refrescar sientope
  noinserta
;

: ^f
  {trans
  tope 1+ fin =
  if
    tope 1+ refrescar sientope
   else
    fin 1+ .my
    -my
    ilfin                    % lfin <- 1
    my refrescar
    noentope
  fi
;

: ^n
  lfin cy tpagina + - dup 1 <=
  if
     drop ^f
    else
      {trans
     .lfin
     tpagina 0 do +my loop
     my refrescar
  fi
;

: ^u
  lfin cy tpagina + + dup #cr? >=
  if
     drop ^t
   else
     {trans
     .lfin
     tpagina 0 do -my loop
     my refrescar
  fi
;
```

```
: ^m                                        % retorno de carro
    {trans                                  % transferir renglon a memoria
    >my
    1 agrandar
    cr myasig
    cy #renglones 1- <> if
      >cy
      +1fin
    else
      noentope
      ^scroll
    fi
    iren                                    % inicializar el renglon
    imx
    insertalinea                            % insertar linea en pantalla
    +#cr

;

completa ^m rcautomatico                    % se usa en mxasig .

: ->sigpalabra  ;

m:: renglon insercion >> caracter posicion &
      1 agrandarbuffer                  % agrandar el renglon
      caracter mxasig                   % insertar caracter en  mx avnza mx y cx
      sicambio
      }pantalla                         % enviar renglon a la pantalla
;

:: texto >> caracter  &
   cx #columnas 1- =
   if
     else
       imodo
       if
        caracter insercion
      else
        caracter echo
        caracter mxasig
      fi
   fi
;


% la funcion siguiente se escribio usando la capacidad de Forth para
% crear funciones primitivas. e.d. en ensamblador Z80.

% El compilador  create  genera un encabezado de una funcion primitiva
% con el nommbre que se le da a continuaciôn. El cuerpo de la funcion
% se mete en lenguaje de maquina.
```

```
nex

create
busqueda    % nombre de la funcion primitiva
%                   ; ¦ delimitador (lbp @) my #bytes
D9 c, %      exx
C1 c, %      pop bc   ; #bytes
E1 c, %      pop hl   ; hl <- my
D1 c, %      pop de   ; de <- (lbp @)
1A c, % bca: ld a,(de)   ; busca caracter
B1ED , %     cpir
2B20 , %     jr   nz,finb  ; busqueda sin exito
F1 c, %      pop  af ; a <- delimitador
D5 c, %      push de  ; ¦ (lbp @) my.nuevo bc.inicial
E5 c, %      push hl  ; hl: dir 2o car de la cadena
C5 c, %      push bc
13 c, % sc:  inc  de
EB c, %      ex   de,hl
BE c, %      cp   (hl)  ;  (lbp @)actual delimitador = ?
EB c, %      ex   de,hl
0928 , %     jr   z,be  ; si : busqueda exitosa
F5 c, %      push af    ; sp <- delimiwtador
1A c, %      ld   a,(de)
A1ED , %     cpi
1420 , %     jr   nz,npb
F1 c, %      pop  af    ; a <- delimitador
F118 , %     jr   sc              ; sigue comparando
%                                 ; busqueda exitosa
E1 c, % be:  pop  hl    ; bc.inicial
A7 c, %      and  a
42ED , %     sbc  hl,bc
23 c, %      inc  hl    ; hl <- ( bc.inicial bc.final - 1+ )
C1 c, %      pop  bc    ; 2o car de la cadena
0B c, %      dec  bc    ; 1er car de la cadena
E3 c, %      ex   (sp),hl          ; ¦ longcadena
13 c, %      inc  de    ; avanza (lbp @) a la sig. cadena
D5 c, %      push de    ; ¦ longcadena (lbp @ sig.cad)
C5 c, %      push bc    ; ¦ longcadena (lbp @ s.cad) 1er.car.cadena
21 c, %
0000 , %      ld   hl,0  ; exito en la busqueda
E5 c, %      push hl    ; ¦ longcadena (lbp @ s.cad) 1er.carcadena 0
0718 , %      jr   finb
F1 c, % npb:pop  af     ; no es palabra buscada
D1 c, %      pop  de    ; desecha inicadena
D1 c, %      pop  de    ;  "  dir 2o carracter
D1 c, %      pop  de    ; de <- (lbp @)
F5 c, %      push af    ; ¦ delimitador
D018 , %     jr   bca   ; continua buscando
D9 c, % finb: exx       ; ¦ delimitador ( busqueda sin exito)
E9FD , %     jp   (iy)
```

```
:: ^i ** car ap dirlbp myinicial caracteres delimitador &
    noinicio
    {trans my .myinicial
    0 #renglones 1- .xycursor
    borralinea inline
    dOd .delimitador              % delimitador <- cr
    lbp @ c@ .car
    lbp .ap
    lbp @ .dirlbp
    car cr =
    if
      lfin cy - .lfin
      my refrescar
      tope my 1- =
      if
        sientope
       else
        noentope
      fi
    else
      delimitador    %  cr
      dirlbp         % (lbp @)
      my             %  my
      fin my -       %  #bytes
      busqueda
      not
      if
                     % ¦ longcadena (lbp @ s.cad) 1er.carcadena 0
        .my          % my <- 1er.carcadena
        drop drop    % tira: longcadena (lbp @ s.cad)
        my 1- c@ cr = if else -my fi
        lfin myinicial my #cr 1+ cy + -
        .lfin
        my refrescar
        noentope
        caracteres fin 1+ !
      else
        myinicial .my       % no la encontro
        my refrescar
        noentope
        caracteres fin 1+ !
      fi
    fi
;
```

```
:: reemplaza  >> v-longcadena lbp@scad lbp@ 1ercar ** n-longcadena &

1ercar my -
renglon + !mx          % longcadena (lbp @ s.cad) (lbp @)

lbp@scad       % dir inicial
50             % longitud 80 caracteres
cr             % byte

cpir

if
    drop v-longcadena .n-longcadena
  else
    1- lbp@scad - .n-longcadena
fi

n-longcadena v-longcadena - dup
0=

if
  drop
 else
  dup
  0<

  if
    minus
    achicarbuffer
   else
    sinserta
    agrandarbuffer
  fi                                          % cadenav<-cadenan

fi

mx                % destino
lbp@scad          % fuente
n-longcadena      % longitud

ldir

sicambio {trans   % actualiza el renglon
```

```
: ^o ** car ap dirlbp myinicial caracteres delimitador 1ercar &

 noinicio
 {trans my .myinicial
 0 #renglones 1- .xycursor
    borralinea inline
    2f2f  .delimitador   % delimitador <- slash
    lbp @ c@ .car
    lbp .ap
    lbp @ .dirlbp
    car cr =

    if
      lfin cy - .lfin
      my refrescar
      tope my 1- =

      if
        sientope
       else
        noentope
      fi

    else

    <do
      delimitador   %  slash
      dirlbp        % (lbp @)
      my            %  my
      fin my -      %  #bytes
      busqueda
      not

      if
        dup                 % ¦ longcadena (lbp @ s.cad) 1er.carcadena 0
        .my .1ercar         % my <- 1er.carcadena    1ercar <- posicion
        my 1- c@ cr = if else -my fi
        }trans
        dirlbp 1ercar reemplaza
        +my
      else
        myinicial .my       % ya no hay mas.
        my refrescar
        noentope
        caracteres fin 1+ !
        return
      fi

    od>

    fi
;
```

```
: ^b 1 achicarbuffer }pantalla sicambio ;
: ^v sinserta ;
: ^p }pantalla ;
: ^d {trans ^ ;
: rubout }trans }pantalla ;
: ^a imx 0 .xcursor ;        % Va a el inicio del archivo
: ^x mx fren mx - dup 0 <=
  if drop drop else aspace fill sicambio }pantalla fi
;
:: ^y ** dir pos &
   fren dup                  % Pone el cursor despues del ultimo caracter
   .dir                      % del renglon.
   renglon - .pos
   pos #columnas 1- =        % Si el renglon esta lleno
   if
     renglon #columnas 2- + .mx  % ponlo en el ultimo caracter visible.
     sicambio
    else
     dir .mx
   fi
   noinserta                 % termina modo insercion
;
:: rlinea >>/dirmy ** dlfin dcy &
   lfin #renglones - .dlfin
   #renglones cy - .dcy
   dlfin 0 <=                          % fin dentro de la pantalla.
   if
     #renglones lfin =
     if                                 % fin en la ultima linea de la pantalla.
       #renglones 1- implfin cy 1- .ycursor
     fi
     return
   fi
   dcy 1 =                             % fin fuera de la pantalla.
   if
      }trans  }pantalla                % cursor en ultima linea de la pantalla.
     else                              % cursor en medio de la pantalla.
      dcy 1- 0 do >my loop }trans      % busca linea que se inserta en fin de
      cy #renglones 1- .cy cy .ycursor }pantalla .cy % la pantalla.
      cy .ycursor dirmy .my }trans
   fi
;
: 1bren
          ?fin not
          if
             fren 1+ renglon - achicar -lfin
             borraren
            else
             fren 1+ renglon - achicar -lfin -my
             borraren -cy
          fi
          my rlinea
;
```

```
: ^z
    {trans % para cuando se desea borrar un renglon que no ha sido
    calcula % actualizado .
    fin my >=
    if
      tope my 1- =
      if
         fin my =
         if
           return
          else
           lren sicambio {trans
           }pantalla lfin cy 1+ = if else incmy +cy }trans fi
         fi
        else
            1bren -#cr
       fi
       cx  .xcursor }trans
    fi
;

% ------ fin codigo comun ---------------

ffff constant topegraficas

topegraficas pxorg - constant #bytesgrf  % #bytesgrf <- ( ffff pxorg - )

0 global topgrf
m: topgrf !topgrf ! ;

0 global topdoc
m: topdoc !topdoc ! ;

0 global mgraficas
m: mgraficas sigraficas 1set ;
m: mgraficas nograficas 0set ;

: gnoesta dp @ type [ ?] cret ;
: geditor                          % ctrl para el editor de graficas
  <do
    atoken graficacion @ search
    if
      number not
      if
        gnoesta
        noprompt
        c1set
      fi
     else
      execute
    fi
  od>
;
```

```
: ^g
  sigraficas
  geditor
;

: guardadescriptores  %  [ GUARDA DESCRIPTORES ]
  topdoc fmem 2 - !          % tope del documento
  topgrf fmem 4 - !         % tope de las graficas
  #cr? fmem 6 - !          % # de renglones
                            % quedan dentro del documento
;

: accesadescriptores  %   [ ACCESA DESCRIPTORES ]
  fmem 2 - @ !topdoc
  fmem 4 - @ !topgrf
  fmem 6 - @ !#cr
;
:: busca/crea << estado &
  trae                      % ¦ estado tipo
   if                       % 1 = ascii  0 = obj
     if                     % 1 = nuevo  0 = viejo
         1 .estado
       else
         fmem 7 - !fin     % salva los descriptores
         accesadescriptores  %
         topdoc !tope        % tope <- principio de documento
         0 .estado
     fi
     else
      [ EL ARCHIVO ES TIPO OBJETO Y NO SE PUEDE EDITAR ]
      abort
   fi
;

:: gviejo ** fin caracteres &  % genera archivo viejo cambia "s" por "v".

  9 archivo .fin
  fin 1+ @  .caracteres
  terminador fin 1+ !       % terminador

  0 archivo                 % 1er caracter del arreglo
  A                         % longitud
  7E                        % caracter "."
  cpir                      % busca un punto

  drop                      % supone que siempre lo encuentra
  76 swap c!                % si lo encontro cambia "s" por "v"
  caracteres fin 1+ !
;
```

```
: grf->txt                    %  [ GRAFICAS -> TEXTO ]

% *******  debe actualizar topdoc y topgrf **********

 fmem 7 -
 fin tope - 1+ -                          % #bytes de el texto y direccion destino
 dup
 1+ !topdoc
 1+ #bytesgrf - !topgrf

 topdoc                                   % destino: topdoc
 topegraficas                             % direccion fuente ( ffffh)
 #bytesgrf                                % # de bytes de la grafica

 modo.p                                   % abre ventana graficas
 lddr
 modo.t

;

: txt<->grf
                        %  [ GRAFICAS <-x-> TEXTO ]
 modo.p

 topegraficas           % destino
 topdoc                 % fuente
 #bytesgrf              % numero de bytes
 lddr

 modo.t
;

: borragraficas         % [ BORRAGRAFICAS ]

  modo.p
  0 pxorg c!

  pxorg 1+            % destino
  pxorg               % fuente
  #bytesgrf           % # de bytes
  ldir

  modo.t
;
```

```
: gtexto << resultado &
 0 .resultado
 mgraficas
 if
     tope my #cr #renglones >=
     if
       ^t
  [ salvar esta pantalla. si: presione S ; no: cualqier otra tecla ] key
        dup c# s = swap c# S = +
        if
          tope 1+ #renglones 0 do }cr 1+ loop 1- !fin
          tope fin #cr dup !#cr .lfin
          grf->txt
          guardadescriptores
          borrapantalla
          borragraficas
          nograficas
          else
            1 .resultado  % no guardar el texto
            nograficas
            ^t
        fi
      else       % salva la pantalla
        tope 1+ #renglones 0 do }cr 1+ loop 1- !fin
        tope fin #cr dup !#cr .lfin
        grf->txt
        guardadescriptores
        borrapantalla
        borragraficas
        nograficas
      fi
    else       % realiza ^q normalmente ( solo texto )
      fmem 7 - fin tope - 1+ - 1+ dup
      !topdoc !topgrf
      guardadescriptores
  fi
;
: ^q borrapantalla
   gtexto                        % tope.doc tope.grf
   if
   else
      r> ( llamada a ^q)
      r> r> ( 1editor deja 2 entradas en rs porque es de ::)
      drop drop drop ( tira las tres r>)
      drop drop ( tira los dos parametros de 1editor)
      qfuente                     % texto
      tope icomp !
      fin 1+ fcomp !
      topgrf pdocumento !         % graficas -> documento
      core definitions
  fi
;
```

```
% botar una direccion del stack de llamadas

:: 1editor ** car 1dp & ( SI SE AGREGAN PARAMETROS MODIFICAR ^q P. F.)
   tope 1+ refrescar
   <do
    key .car
    car 7f = if
      rubout
    else
      car 20 <      % si es menor que 20 es caracter de ctrl
      if
        dp @ .1dp
        2 c,
        c# ^ c,
        car         % lee el caracter de  control, hay que prenderle un bit
        40 or       % , el 6 , para convertirlo a caracter imprimible
        c,
        1dp dp !
        context @ @  % vocabulario
        search
        not
        if
          execute
        fi
       else
         car texto
         sicambio
       fi
     fi
   od>
;

core definitions
editando

: edita
  editando
  busca/crea                  % 1 = nuevo  0 = viejo
   if
      imem 1editor            % crealo
     else
      topdoc topgrf <>              % es archivo de graficas ?
      if
          txt<->grf             % si : separa texto de graficas
      fi
      tfuente
      #cr? .lfin
      1editor                           % edita el texto.
   fi
;
```

```
: ^e editando (  tfuente #cr? .lfin)
  topdoc topgrf <>
  if
    txt<->grf
  fi
  tfuente
  #cr? .lfin
  1editor
;

core definitions

eof
```