17
2ej

Universidad Nacional Autónoma de México

FACULTAD DE INGENIERIA

"EDITOR DIRIGIDO POR SINTAXIS"

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A N:

Edgar Hernández Taja

Carlos Mateos Barrera

Alberto Sánchez Pérez

Asesor Ing. Lino Diaz Bello

MEXICO, D. F. 1988

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

I N D I C E .

I. INTRODUCCION ... 1

I.1. Planteamiento de la problemática ... 1

I.2. Definición de objetivos ... 4

I.3. Metodología de desarrollo ... 5

II. ANTECEDENTES ... 6

III. ANALISIS ... 10

III.1. Evaluación de editores de sintaxis ... 10

III.1.1. CORNELL ... 10

III.1.2. SUPPORT ... 12

III.1.3. MAGPIE ... 13

III.1.4. MENTOR ... 15

III.1.5. SAGA ... 16

III.1.6. POE ... 18

III.1.7. YALE ... 19

III.1.8. IPE ... 20

III.1.9. RPDE ... 23

III.1.10. DICE ... 24

III.2. Contrastación de funciones ... 26

IV. DISEÑO ... 32

IV.1. Principios de diseño ... 32

IV.2. Modelo conceptual del EDS ... 34

IV.3. Diseño modular del EDS ... 37
IV.3.1. Procesador de texto ... 37
IV.3.1.1. Captura de variables y expresiones ... 38
IV.3.1.2. Generador de símbolos ... 39
IV.3.1.3. Escritura de texto ... 40
IV.3.2. Manejo de ventanas ... 40
IV.3.2.1. Información de ventanas ... 41
IV.3.2.2. Creación, cálculo y despliegue ... 41
IV.3.2.3. Ventanas de dirección de sintaxis ... 43
IV.3.2.4. Ventanas de comandos de edición ... 43
IV.3.3. Analizador léxico sintáctico ... 43
IV.3.3.1. Identificador de variables y expresiones ... 45
IV.3.3.2. Analizador léxico ... 45
IV.3.3.3. Manejador de tablas ... 47
IV.3.3.4. Manejador de errores ... 47
IV.3.3.5. Analizador de expresiones ... 47
IV.3.4. Estructura interna ... 48
IV.3.5. Comandos de edición ... 49
IV.3.5.1. Comandos de movimiento ... 52
IV.3.5.2. Comandos de modificación ... 52
IV.3.5.3. Comandos de búsqueda ... 52
IV.3.5.4. Operaciones en disco ... 53

V. IMPLEMENTACION ... 54
V.1. Técnicas y herramientas ... 54
V.1.1. Técnica de manejo de ventanas ... 54
V.1.2. Técnicas de edición y manejo de estructura ... 58
V.1.2.1. Modelo de tabla de líneas ... 59
V.1.2.2. Modelo de buffer fijo ... 60
V.1.2.3. Modelo de lista ligada ... 63
V.1.3. Técnicas de implementación de un analizador sintáctico ... 64
V.1.4. Técnica de manejo de tablas ... 72
V.1.5. Técnica de detección de errores ... 77
V.1.6. Nombres de comandos ... 79
V.2. Definición del medio ambiente inicial ... 82
V.3. Descripción del lenguaje a implementar, Pascal ... 84
V.4. Variaciones con respecto al lenguaje Pascal estándar ... 89
V.5. Selección y justificación del lenguaje de implementación ... 91
V.6. Implementación del procesador de texto ... 93
V.7. Implementación del manejador de ventanas... 96
V.7.1. Información de las opciones de las ventanas ... 97
V.7.2. Creación, cálculo y control ... 97
V.7.3. Ventanas de dirección de sintaxis... 97
V.7.4. Ventanas de comandos de edición ... 101
V.8. Implementación del analizador sintáctico... 104

V.8.1. Identificación de variables y expresiones ... 104
V.8.2. Analizador léxico ... 105
V.8.3. Manejador de tablas ... 108
V.8.4. Manejador de errores ... 111
V.8.5. Analizador de expresiones ... 111
V.9. Implementación del manejador de estructura interna ... 114
V.10. Implementación del manejador de comandos ... 120
V.10.1 Comandos de movimiento ... 120
V.10.2 Comandos de edición ... 123
V.10.3 Comandos de búsqueda ... 125
V.10.4 Operaciones en disco ... 125

VI. Evaluación del EDS ... 128
VI.1 Ejemplo prototipo ... 128
VI.2 Resultados de la evaluación ... 139
VII. Conclusiones ... 141

Glosario.
Bibliografía.
Apéndice A.
Diagramas de Conway.
Apéndice B.
Autómatas.
Apéndice C.
Tablas.

INDICE DE FIGURAS.

Nombre	Número	Pág.
Estructura interna de YALE.	... III.1.	21
Ventana de bloque en RPDE.	... III.2.	24
Tabla de contrastación de editores.	... III.3.	27
Modelo conceptual del EDS.	... IV.1.	36
Procesador de texto.	... IV.2.	39
Manejador de ventanas.	... IV.3.	42
Analizador sintáctico.	... IV.4.	46
Manejador de estructura interna.	... IV.5.	50
Manejador de comandos de edición.	... IV.6.	52
Modelo de tabla de líneas.	... V.1.	60
Modelo de buffer fijo.	... V.2.	63
Modelo de lista ligada.	... V.3.	65
Flujo de ventanas de sintaxis.	... V.4.	99
Ventanas de comandos de edición.	... V.5.	102
Estructura interna de línea de texto.	... V.6.	116
Liga de inicio y fin de template.	... V.7.	118
Presentación del EDS.	... VI.1.	130
Menú principal.	... VI.2.	130
Ventana de edición.	... VI.3.	131
Ventana de declaración.	... VI.4.	132
Declaración de variables.	... VI.5.	132
Declaración de tipos.	... VI.6.	133

Nombre	Número	Pág.
Ventana de tipos simples.	... VI.7.	134
Variables ya declaradas.	... VI.8.	134
Ventana de instrucciones.	... VI.9.	135
Variables no declaradas.	... VI.10.	136
Instrucción For...to...do	... VI.11.	137
Expresión incompleta.	... VI.12.	137
Falta de parentésis en una expresión.	... VI.13.	138
Edición terminada.	... VI.14.	138

I. INTRODUCCION.

I.1 Planteamiento de la problemática.

Atendiendo a la necesidad de los medios actuales de programación de crear nuevas herramientas que permitan una interacción con el usuario mas rápida, amigable y eficiente, surgen los llamados editores de sintaxis o editores de lenguaje que, incluso, proporcionan muchas más ventajas para el programador, las cuales se irán mencionando conforme se conozca mejor su objetivo y funcionamiento.

Un editor de sintaxis como tal es un término poco conocido, por lo que conviene definir claramente en que consiste; para esto es necesario definir por separado los conceptos de editor y de sintaxis.

En cuanto a la sintaxis, desde el punto de vista de la lengua, se define como el estudio de las relaciones que se establecen entre las palabras para formar una oración. Adaptando esta definición al lenguaje de programación, podemos decir que la sintaxis es la relación que deben guardar las palabras y los símbolos, para establecer una declaración, expresión o procedimiento. Sin embargo estas relaciones no pueden ser descritas libres del contexto de la gramática por lo que, de un modo mas formal, la definición suele dividirse en dos partes:

Una parte es especificada por la gramática libre de

contexto, que describe la estructura permitida por el lenguaje de programación , y la otra, que consiste en restricciones colocadas en la sintaxis libre de contexto, estas últimas incluyen reglas de intercalamiento de tipos para identificadores y el requerimiento de que la llamada de un procedimiento contiene exactamente los mismos argumentos que parámetros en la definición.

Por otro lado, un editor es una herramienta de los medios de programación usada como camino inicial de entrada y subsecuentemente en la modificación de programas desarrollados en un lenguaje determinado. Un editor es en si un manipulador de texto de propósito general y usualmente no se sabe si la entrada es un programa ó un documento. Programas de éste tipo se concibieron para escribir, o corregir el texto del que se deriva un programa (código fuente), pero actualmente han tenido un notable desarrollo y constituyen una de las aplicaciones más útiles e interesantes de las computadoras. Un editor permite desarrollar funciones tales como: escritura de un texto, conservación y corrección del mismo, anulación o inserción de partes y cambio automático de caracteres, palabras o frases enteras.

Con lo anterior podemos definir de una manera mas clara y entendible lo que es un editor de sintaxis; que consistirá en un editor que pierda la característica de ignorar si la entrada es un programa o no, sabiendo específicamente que sí

lo es, y que además deberá asegurar una sintaxis correcta del lenguaje en el que esta escrito, sin tratar con la semántica ni con las producciones dependientes del contexto. Lo anterior implica que el editor como tal estará cumpliendo simplemente las funciones básicas de edición, pero además complementadas por la realización de un chequeo sintáctico, que es una de las fases de un compilador. El editor proveerá entonces, de todas las palabras clave necesarias, así como separadores, terminaciones y todo lo sintácticamente requerido por el lenguaje, desapareciendo algunas ambiguedades debidas a una construcción mas clara de aproximación al programador.

De un estudio empírico de la Universidad de Carnegie-Mellon en Pittsburg[1] para detectar los errores de programación mas comunes, se encontró que para cientos de posibles errores-advertencias encontradas, en alrededor del 90% de los estudiantes incurrían en los siguientes errores:

a) Variable ya declarada.
b) Variable no declarada.
c) Variable no inicializada.
d) Signos de puntuación faltantes.
e) Tipo erroneo o faltante.

De lo anterior se puede ver que un editor de sintaxis puede eliminar la mayoría de éste tipo de errores y muchos otros evitando recompilaciones innecesarias.

Editores de éste tipo han sido desarrollados en Universidades de los Estados Unidos desde 1971, alcanzando un alto grado de desarrollo a nivel experimental y, como parte de medios ambientes con enfoques muy específicos. En México, el editor de sintaxis es poco conocido y no se tiene noticia de su desarrollo.

Lo anterior, junto con las ventajas que ofrece un editor de sintaxis, es, entre otras, una de las motivaciones principales de éste proyecto.

I.2 Definición de objetivos.

A partir de lo anterior, y enfocandonos en lo que definimos como editor de sintaxis, se plantean los siguientes objetivos a cubrir en el presente trabajo.

1.2.1. Ofrecer una herramienta poderosa para una edición rápida, amigable y eficiente de programas escritos en lenguaje Pascal.

1.2.2. Obtener un editor de fácil uso y aprendizaje que pueda apoyar el conocimiento del lenguaje Pascal.

1.2.3. Elaborar un editor que permita garantizar la

creación de programas escritos en lenguaje Pascal libres de errores léxicos y sintácticos.

1.2.4. Proporcionar al editor la facilidad de modificar programas escritos en lenguaje Pascal libres de errores de sintaxis,sin alterar esta característica.

I.3 Metodología de desarrollo.

A partir de los editores de sintaxis desarrollados en otras universidades, conociendo sus características principales y contrastándolas entre sí, se plantea una metodología de desarrollo de tipo deductivo partiendo de los editores existentes hacia un planteamiento personal mas particular, replanteándose éste cuantas veces sea necesario.

II. ANTECEDENTES.

Desde el inicio de la computación, siempre ha existido la necesidad de introducir de una manera mas rápida y eficiente los programas de aplicación. Como se sabe, en las primeras grandes máquinas, los programas eran introducidos por medio de una serie de interruptores que representaban en combinación ciertas instrucciones. En ello se ocupaba mucho tiempo, y las correcciones eran sumamente complicadas, ya que no solo había que conocer los códigos de las instrucciones, sino también las direcciones absolutas de la memoria de todos los datos. Después de ello, aparecen muy rápidamente los lenguajes simbólicos, que, en general, eran mnemotécnicos. El uso de estos lenguajes obliga a decodificarlos mediante programas traductores o ensambladores que dan origen al software traductor.

Con la segunda generación de computadoras aparecen los ensambladores básicos y posteriormente los compiladores. Muchas rutinas son útiles a los usuarios para las operaciones mas elementales (copias de información, clasificación de archivos, etc), también aparecen los programas de explotación que son programas que permiten el encadenamiento rápido de los programas usuario.

En la tercera generación aparecen los primeros Sistemas Operativos para las computadoras y aparecen nuevos lenguajes, proporcionando un gran avance en el desarrollo versátil

de nuevas aplicaciones.

Es el lenguaje de alto nivel el que permite elaborar programas de aplicación mas especificos, junto con las rutinas de utileria e incluyendo aquellos que permiten introducir y corregir programas para la máquina. Nos referimos a los editores, que se desarrollarán con gran rápidez y eficacia.

En primera instancia surgen los editores de texto, que permiten editar cualquier clase de programas por medio de comandos y guardando la información sin importar de lo que se trata, es decir, en forma de una simple secuencia de caracteres. Mas tarde se desarrollan editores de texto que son capaces de formatear un documento, permitiendo mover bloques del mismo, sustituirlos, copiarlos e incluso centrarlos, esto trae a la larga el desarrollo de los llamados procesadores de palabras que han tenido un gran auge en la actualidad.

También, aunque un poco más tarde, se logra editar en pantalla figuras gráficas, que con el aumento de la resolución de los monitores, se permite en la actualidad la edición de gráficas con una gran gama de posibilidades tales como rotación de figuras, aumentos y disminuciones de estas, coloreado, etc.

Los medios actuales de programación incluyen entre sus múltiples herramientas que incrementan la productividad de la programación, facilidades de edición en muchas variedades asi como en la compilación, ligado, interpretación, ejecución, actualización, mantenimiento, etc. Esta tendencia ha marcado tambien, el desarrollo de editores por separado de los medios de programación, siendo herramientas con múltiples enfoques cada día mas sofisticados.

Siendo considerada como una herramienta sumamente importante para el desarrollo eficiente y rápido de los sistemas de información, los editores estructurados u orientados a lenguaje inician su vertiginosa carrera en 1971 con un sistema sofisticado de edición llamado EMILY [2], en donde programas en PL/I pueden ser creados, visualizados y modificados en términos de estructuras sintácticas del lenguaje, lo cual marca un hito en el desarrollo de los editores que de este tipo son llamados editores estructurados, orientados a lenguaje o una de sus especialidades "Editores Dirigidos por Sintaxis (EDS) ".

Sin embargo, el primer editor de este tipo en tener éxito aplicable, es desarrollado en la Universidad de Cornell y se utiliza desde 1979 con fines de enseñanza, este editor es parte de un medio ambiente de programación, el Sintetizador Cornell [3].

Mas recientemente, el proyecto GANDALF (1982) [4] y el

SAGA (1984) [5] , han hecho investigaciones prácticas en medios ambientes de programación que soporten una ayuda conjunta en todos los módulos que los componen.

El desarrollo de este tipo de editores no se detiene, llegando a incluir nuevas innovaciones como el uso de gráficos y ventanas en la edición como el editor RPDE [6], ó ideogramas como el IDEOSY [7]. Se pretende entonces, que la gran atención puesta en el desarollo de editores estructurados, será básica para la nueva generación de usuarios, siendo mas potentes y flexibles, que producirán el desplazamiento de los editores tradicionales de texto.

III. ANALISIS.

III.1 Evaluación de editores de Sintaxis.

El punto fundamental del análisis, consiste en una contrastación de los diferentes editores existentes para el planteamiento de un diseño propio. Para ello se aplica una metodología de tipo deductivo desarrollandose de lo general (editores existentes) a lo particular (EDS).

Para lo anterior, se presenta a continuación, un resumen de los editores de sintaxis más conocidos y flexibles existentes, que nos permitirán una futura comparación para el planteamiento del EDS.

III.1.1. CORNELL.

El programa sintetizador Cornell [3] es en realidad un medio ambiente de programación cuyo punto medular es un editor dirigido por sintaxis, en donde la entrada y modificación de textos es guiada por la gramática del lenguaje de programación, las características principales de este editor se resumen a continuación:

- Todos los tipos de instrucciones son definidos como templates, que proporciona las palabras clave y los signos de puntuación, incluyendo huecos en donde es necesaria la edición de código.

-Los programas son creados en forma TOP-DOWN insertando nuevos templates o frases dentro del esqueleto del template previo, sólo los huecos dentro del template pueden ser expandidos.

-Los templates son generados por comandos, aunque las expresiones son tecleadas por el usuario.

-Los movimientos del cursor en la pantalla corresponden a un recorrido transversal preorder de un árbol de sintaxis abstracta.

-El cursor no aparece en las palabras reservadas, sino que se mueve de hueco en hueco y de template en template, sólo en lugares donde la inserción, supresión y cambios son permitidos.

- El sintetizador asegura que solo unidades sintácticamente completas pueden ser modificadas.

- La ejecución del programa es suspendida cuando se encuentra un hueco no expandido regresando el control al editor con el cursor posicionado en ese hueco.

- Los archivos estan representados internamente por árboles derivados. Cada template o frase esta representado en el árbol por nodos separados.

- La representación de despliegue es generada a partir del árbol, la representación del texto no es almacenada en el árbol.

III.1.2. SUPPORT.

Los trabajos sobre el medio ambiente de programación SUPPORT [8] fueron iniciados en 1981 por el National Bureau of Standards y continuados en 1982 por el NBS y la Universidad de Maryland. SUPPORT fue implementado en Pascal para una VAX11/780 bajo el sistema operativo UNIX.

Las principales características de su editor se resumen a continuación.

- Utiliza una representación interna de árbol.
- Emplea el manejo de ventanas.
- Aplica el concepto de Buttons, teclas especificas del teclado con la finalidad de editar las estructuras de las instrucciones.
- Permite el manejo o edición de programas de hasta 20,000 lineas de código fuente.
- Maneja comandos para insertar información directamente en la estructura del árbol.
- Posee comandos para salvar el programa en un archivo.
- Genera una versión del programa en código de diseño y otra en el código fuente de pascal.
- El programa en representación interna de árbol puede ser copiado a otro diskette.
- Utiliza un parser híbrido que proporciona la opción de crear el programa en dos formas:

- TOP DOWN

- BOTTOM UP

- Cada nodo del árbol posee la información necesaria para desplegar su contenido textual en el video.

- Proporciona comandos para realizar una edición textual.

- La sintaxis verificada por el editor es determinada por la gramática de Pascal.

- Permite la integración de programas no creados con este editor a él, para poder modificarlos.

- La edición textual es tratada por el parser como BOTTOM UP.

- La edición auxiliada por los Buttons es tratada por el parser como TOP DOWN.

- El parser utilizado es un LALR híbrido.

III.1.3. MAGPIE.

Magpie [9] es un medio ambiente de programación que permite checar reglas de sintaxis y semántica al tiempo de la edición, además proporciona comandos dentro del contexto de la ventana para la edición y ejecución del programa en código fuente de Pascal. En MAGPIE el usuario no necesita tener control sobre el inicio del proceso de compilación, ya que este proceso no modifica o actualiza el código objeto generado si detecta errores, y además es realizado en su última parte al

ejecutar el programa, debido a la previa precompilación realizada durante su edición.

Las características principales de edición dentro de este medio de programación se resumen a continuación:

- Ejecuta un análisis de sintaxis y semántica al tiempo de la edición.
- Se auxilia de un conjunto de ventanas para la edición.
- Dentro de cada ventana se despliega un menú, el cual corresponde a la finalidad o actividades que pueden realizarse en esa ventana.
- Cuenta con un conjunto de comandos para la edición textual.
- Maneja el concepto de áreas de trabajo:

1.- CODE BROWSERS, esta área permite editar y modificar el programa deseado.

2.- DEMON BROWSER, área para la ejecución de los programas.

3.- WORKSPACE, área de trabajo para debugger.

4.- STACK BROWSER, estado del programa durante su ejecución.

- Las ventanas son generadas por el usuario dentro de las áreas de trabajo anteriores, en cada área se pueden crear mas de una ventana.
- El chequeo de la sintaxis esta basada en la gramática de Pascal.

- Utiliza un compilador incremental para el análisis léxico.

- La representación interna del programa editado se basa en una tabla de símbolos, y un árbol que contiene los atributos y tipo de las instrucciones y datos del programa.

- El chequeo sintáctico se lleva a cabo en los intervalos en que el sistema no esta disponible para la edición, después de un ENTER.

- Técnicas para un parser incremental son utilizadas para la construcción y recorrido de la estructura de árbol.

III.1.4. MENTOR.

Mentor [10] es un sistema usado para manipular información estructurada, pero lo que se intenta es crear un ambiente de programación en el cual el programador pueda diseñar, implementar, documentar, depurar, probar, validar, y mantener además de poder transportar sus programas. El primer paso fue diseñar un editor estructurado para Pascal, implementado en MENTOR-PASCAL. Después, se continuó para poder implementar herramientas de normalización y documentación de programas en Pascal. Otro punto fue el depurar el lenguaje fuente.

Las características de edición principales de este sistema son las siguientes:

- MENTOR es un procesador diseñado para manipular estructuras de datos.

- Los datos manipulados por MENTOR estan representados por árboles de sintaxis abstracta.

- La comunicación se hace a través de un intérprete especializado en manipulación de árboles en lenguaje MENTOL.

- La sintaxis abstracta es representada por medio de un árbol para la creación de un programa estructurado.

- Los operadores son los bloques básicos, construidos en el lenguaje.

III.1.5. SAGA.

El Proyecto SAGA [5] (Software Automation Generator and Administrator) investiga los problemas del manejo del ciclo de vida del software en proyectos de desarrollo de pequeño y mediano tamaño. El software puede ser desarrollado para una aplicación especifica que debe ser confiable en experimentos que requieren de prototipos rápidos. El sistema debe ser flexible y acomodarse a varios lenguajes de programación. Una de las principales herramientas del sistema y tema que nos ocupa, es el editor orientado a lenguaje de SAGA, que combina muchos de los mejores rasgos de los mejores editores de este tipo, a continuación se resumen sus principales características.

- El editor es la interfase de SAGA. se construye por medio de un editor generador. Asociado al editor hay varios modulos de evaluación semántica y sintáctica, versión de control, documentación y análisis.

- El editor almacena texto en forma de árbol de parser y tabla de símbolos.

- El editor utiliza comandos de edición para construir el programa mientras se soporta con comandos orientados a lenguaje.

- Las cadenas de texto son generadas por rutinas de listas doblemente ligadas de nodos terminales.

- El editor esta orientado a pantalla; posicionando el cursor en cualquier punto del texto en la pantalla, se puede insertar, reemplazar o borrar texto directamente.

-Los comandos de edición permiten insertar, borrar, reemplazar, mover y compilar texto.

-El árbol de parser se implementa con un archivo de páginas que contiene nodos. Esta estructura de datos se pagina en memoria por demanda. Solo las páginas requeridas por el editor se encuentran en memoria.

- El manejador de despliegue del editor, manejador de árbol, rutinas de soporte, rutinas de interpretación de comandos y la ejecución de comandos no necesitan modificación.

- La entrada al generador de parser consiste en una descripción formal de la sintaxis del lenguaje en forma de gramática, una descripción formal de las representaciones léxicas de los tokens y evaluación de la semántica en forma de código ejecutable.

III.1.6. POE.

POE [11] (Pascal Oriented Language) es un editor basado en lenguaje de pantalla que reconoce reglas semánticas y sintácticas de pascal. Su diseño comenzó en 1979 y su primera versión fue para VAX de serie 11 y en HP9800. Esta escrito en pascal y su diseño permite ser fácilmente transportable a otras máquinas. Esta inspirado en gran parte por el Cornell Sinthesizer aunque fisicamente es más parecido al sistema COPE [12] . Aqui se muestran algunas de sus características y funciones.

- Esta orientado a estructura y desarrollo de modelos por medio de prototipos de lenguaje, no usando templates.
- En la interfase, el usuario mueve el cursor a un promt desplegando el texto correspondiente. Escribiendo un token se produce una expansión particular para éste.
- Existen 3 tipos de símbolos:
  - -Huecos que serán expandidos para la obtención de un programa.
  - -Huecos que producirán una construcción.
  - -Símbolos ordinarios.
- No se pueden crear estructuras semánticamente incorrectas.
- Se tiene un algoritmo de corrección de errores.
- Se opera el texto por medio de archivos, tiene la desventaja que incrementa el tiempo en el inicio u edición.

- Utiliza una aproximación al parser top-down.

- El parser utiliza manejo de tabla LL con una provisión especial hecha por listas.

- Los árboles se construyen con atributos que mas tarde serán evaluados.

- Es un editor orientado a estructura y no a caracter por lo que carece de comandos de búsqueda.

III.1.7. YALE.

YALE [13] es un editor orientado a lenguaje enfocado a la información semántica consistente, cuya filosofía es garantizar la sintáxis correcta del programa. Aquí se presentan sus principales características.

-Emplea atributos gramáticos en forma de lenguaje independiente.

- Asegura una sintaxis correcta pero no trata con semántica ni información dependiente de contexto, no incrementa la compilación ni la depuración.

- Existe un programa llamado Translator, donde la información semántica es procesada produciendo código objeto.

- Explota la idea de Wirth [14] para la construcción del parser.

- Las estructuras de datos pueden ser fácilmente modificadas en un sistema tubular de templates que corresponden a producciones de gramática.

- El sistema de templates generado representa una sintaxis concreta del lenguaje.

- Los templates generados son catalogados en un archivo y entonces el editor los lee cuando se usan.

- La edición se reduce a escoger el template con el cursor en el menú.

- Cuando el cursor se mueve a cualquier no terminal, el menú debe suministrar otra lista de templates en ejecución top-down.

- Se guarda una lista de items borrados de acuerdo al tipo de sintáxis por si se desea recuperarlos.

- Con YALE no se incurre a overheads en el almacenamiento del programa como código intermedio como lo hacen los editores convencionales que almacenan el programa como texto.

- El editor trata las expresiones como texto lineal.

- Utiliza un árbol de tipo binario que crece en dirección del nodo seleccionado por el cursor. Un ejemplo del manejo de éste árbol se presenta en la figura III.1.

III.1.8. IPE.

IPE [15] es un medio de programación basado en la tecnología de compilación, que provee facultades sólo encontradas en los intérpretes. El programa es manipulado por un editor dirigido por sintaxis y su ejecución es controlada por un depurador integrado al editor. A continuación se describen las principales características de su editor.

```
BEGIN
 X:=3;
 Y:=2;
 IF P=Q THEN
    BEGIN
      Z:=1;
      R:= Z*(X+Y);
    END;
END
```

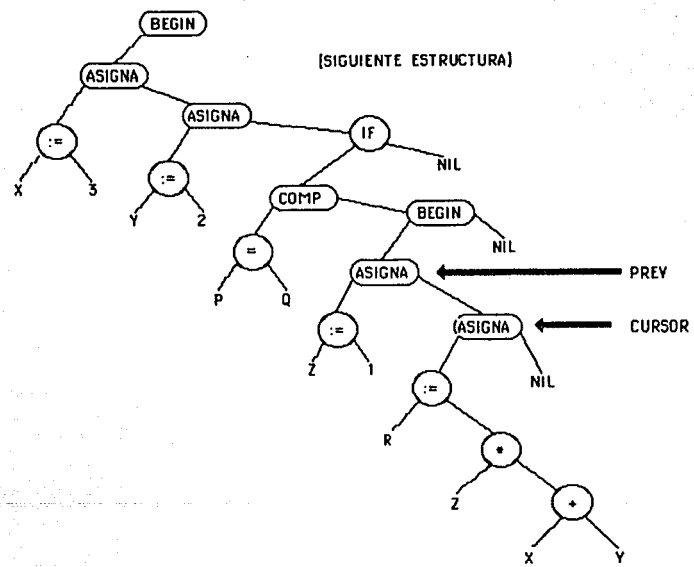


FIG. III.1 ESTRUCTURA INTERNA DE YALE

- El editor construye y manipula directamente un árbol sintáctico del programa.

- El programador construye el programa insertando templates que representan diferentes construcciones del lenguaje y después llena los huecos con otros templates.

- El editor provee todas las palabras clave necesarias, asi como separadores, terminales, y todo lo sintácticamente requerido por el lenguaje.

- El programador interactúa con el editor a través de comandos de lenguaje que son usados para construir nuevos templates y comandos de edición para manipular el árbol del programa.

- El editor invoca rutinas de chequeo de semántica mientras el programador construye o modifica éste.

- Cuando se termina la edición, el programa es sintáctica y semánticamente checado aunque no este completo. No se produce código hasta que no este sintácticamente correcto.

- En el árbol los terminales son usados para representar variables, constantes y elementos estáticos (tipos), además de construcciones expandibles.

- El nodo de variables contiene información de la tabla de símbolos, provee de espacio a información semántica como el tipo de variable y referencias de otras ocurrencias de la misma.

- Los no-terminales describen subárboles del programa correspondientes a construcciones de flujo de control y definiciones de datos. La información del nodo incluye el tipo de construcción, referencia del nodo emparentado y su padre.

- Las correcciones de semántica no son forzadas (se pueden usar variables antes de ser declaradas).

III.1.9. RPDE.

RPDE [6] es un medio de programación que provee de todas las funciones propias de una estación de trabajo de alto rendimiento. El corazón del medio ambiente es un editor que despliega el material (código, documentación, especificación u otra información) en dos dimensiones en forma gráfica. Este editor estrucurado previene al usuario de cometer muchos errores comúnes de sintaxis. Las características fundamentales que lo hacen diferente de los anteriores editores estructurados son:

- Utiliza las facilidades de los gráficos en las partes que requieren de bloques intercalados.
- Provee de "templates" genéricos para modularización, repetición, condición y declaración.
- Despliega comentarios en video inverso como parte asociada del bloque de código.
- El cuerpo del programa consiste sólo de bloques convencionales con 3 condiciones y sus correspondientes acciones.
- No existen inicios de bloques (begin), ni fin de bloque (end), sino que la estructura de cada bloque es proporcionada por los gráficos por medio de ventanas (fig III.2).

- En el despliegue se seleccionan los objetos más importantes y los coloca en la pantalla, indicando con puntos suspensivos donde se omitió código.

- RPDE genera código para IBM PC de Pascal compatiable.

- Proporciona selección de comandos para insertar, borrar y modificar texto.

NOMBRE TABLA DE SIMBOLOS LOCAL

Simple	Variable I : integer
Descripción del bloque (comentario)	
Procedure Var a:integer	
a := a + i	

Fig. III.2 Ventana de bloque en RPDE.

III.1.10. DICE.

El sistema DICE [16] es un medio ambiente de programación altamente integrado que proporciona al programador soporte en medios ambientes residentes en el host. El sistema contiene herramientas tales como un editor con estructura orientada a pantalla, un compilador incremental, un depurador orientado a pantalla y una base de datos. Para nuestro interés mencionaremos algunas características del editor orientado a pantalla.

- Los editores orientados a lenguaje por éste sistema, son editores estructurados y textos híbridos que proporcionan una máxima flexiblilidad.

- El parser es automáticamente ejecutado de acuerdo a la subgramática correspondiente, es decir, a la categoría sintáctica de un determinado subárbol.

- El diseño esta influenciado por editores híbridos del sistema PATHCAL [17] y el Cornell Program Sinthesizer, junto con la experiencia obtenida por los editores de árbol de INTERLISP [18].

- Los programas en el medio ambiente DICE son internamente representados como árboles de sintáxis abstracta.

-La edición del programa se logra con un editor de estructura de pantalla que marca los nodos nuevos o cambiados en el árbol del procedimiento en turno.

- La recompilación de nodos cambiados se ejecuta durante el recorrido preorder del árbol.

- La edición e impresión puede ser generada de las especificaciones de sintáxis argumentadas con la edición y las acciones de impresión.

- El parser, la edición y la impresión son manipulados por tablas que proporcionan un alto grado de independencia con el lenguaje.

- Los comentarios son representados como nodos especiales en la sintaxis abstracta y pueden aparecer directamente alrededor de instrucciones o declaraciones.

III.2 Contrastación de funciones.

Para la contrastación mencionada se presenta en la siguiente hoja una tabla comparativa, que nos facilitará el análisis (figura III.3).

Como podemos ver, existen varios puntos en común en los cuales nos enfocaremos inicialmente. En primer lugar, más de la mitad de los editores implementados, editan programas en Pascal, lo cual se debe en gran parte a su facilidad de plan teamiento en su gramática estructurada y a su gran auge en los ultimos años. Esto nos afirma, aunado al innegable primer sitio que ocupa en la enseñanza de la carrera de Ingeniero en Computación de la UNAM, la necesidad de que el EDS sea desarrollado para lenguaje Pascal.

De acuerdo a la descripción de los editores podemos ver que gran parte de ellos pertenecen a medios ambientes de programación y aún en estos casos representan el corazón de sus ambientes, otros como el POE, YALE y RPDE son diseñados solo como editores. Sin embargo en todos ellos podemos ver que el seguimiento de la gramática del lenguaje en turno proporciona el punto medular del análisis sintáctico.

Para ello, todos a excepción del POE utilizan los llamados templates, que son unidades sintácticamente correctas, que estan predefinidas, y que eliminan tareas mundanas en el desarrollo del programa. Su inserción es económica debido al

TABLA COMPARATIVA DE EDITORES DIRIGIDOS POR SINTAXIS.

EDITOR	AUTOR	REP. INTERNA	LENGUAJE	INT. USUARIO	ANAL. EXPRES.	ANAL. ERRORES	CARAC. ESPECIALES
YALE	R. Zavednik	Arbol Sintáctico binario que crece hacia donde crece el cursor.	Contexto libre Notación BSI.	Sencilla, sin manipulación de formatos.	En forma de texto	----------	Es pequeño y su cambio de programación es fácil.
IPE	Medina Mora	Arbol Sintáctico y representación de máquina de TCOL-ADA.	Algol ADA GC	Muy sencilla con ayuda de comandos.	----------	Solo avisa que detecta un error.	Modularización y abstracción de datos que lo hacen eficiente.
CORNELL	Teitelbaum	Arbol Sintáctico derivado.	PL/CS PL/1	Fácil y versátil.	Texto tecleado por el usuario.	Exige su corrección al momento.	Crea un ambiente de programación.
MENTOR	Veronique Donzeau Gouge, Gerard Huet.	Arbol de sintaxis abstracta.	Pascal.	Sencilla.	Como parte de la estructura del árbol.	Se analiza al final como compilador.	Da mantenimiento y transporte a los programas.
SAGA	R. CAmpbell P. Kirslis	Arbol de Parser. Cadenas de texto en listas dobles.	Varios.	Comandos de edición y de lenguaje.	Como texto o como arbol	Solo marca los error.	Modularización diseñado para experimentos con prototipos rápidos.
POE	C. Fisher G. Johnson.	Listas doblemente ligadas y arboles.	Pascal.	Por medio de despliegue de PROMTS.	Como texto.	No desaparece hace su corrección.	Facilmente transportable a otras máquinas.
RPDE	W. Harrison J. Rosenfeld.	----------	Pascal.	Sencilla, por medio de comandos.	Como texto modificable.	----------	Despliegue gráfico e impresión compactada.
DICE	Peter Fritzon	Arbol Sintáctico abstracto.	Pascal Fortran.	Con comandos.	Como texto.	----------	Es un editor estructurado y de texto hibrido.
SUPPORT	M. Zelkowitz.	Arbol Sintáctico	Pascal.	Teclas de funciones y comandos.	----------	----------	Utiliza manejo de ventanas. Puede integrar programas de otro editor
MAGPIE	N. Doglie D. Menicosy M. Swartz.	Tablas y arboles de atributos.	Pascal.	Comandos de edición textual.	En forma de texto.	----------	Se auxilia de ventanas de selección

Fig. III.3.

tiempo de creación y corrección. Además de minimizar errores al teclearlo. Por esto la decisión de utilizar templates en el desarrollo del EDS es ineludible, y el tratamiento que se debe dar a estos dependerá de la orientación fundamental del editor.

Por otro lado en cuanto a la interfase con el usuario, vemos que se trata de un conjunto de comandos de edición, apoyados por otros comandos de ayuda del lenguaje haciendolas sencillas pero teniendo el inconveniente por antonomasia de que el uso de comandos requiere memorizarlos para una edición eficiente. Existe el caso especial de SUPPORT que programa teclas de funciones para los comandos mas utilizados, o el caso de POE que despliega PROMTS para ser manipulados o bien el caso de MAGPIE que se auxilia de ventanas para la edición. Existe también un punto importante en el manejo de ventanas, en YALE que despliega los templates permisibles en el hueco posicionado, o bien uno de los más interesantes que lo representa RPDE, en donde las ventanas encierran al template junto con sus principales caracteristicas.

Tratando de buscar una interfase, sencilla como en todos los editores, pero a su vez eliminando los problemas del uso de comandos, se puede optar por un manejo de ventanas que genere los templates permisibles al momento de la edición como YALE, pero que a su vez también incluya comandos de edición como MAGPIE. Algo similar a la versión de SUPPORT pero con un

enfoque mas dinámico y sencillo.

Si observamos ahora el tipo de representación que manejan internamente los editores, encontramos que, todos a excepción de POE, utilizan árboles sintácticos, la manera en que los utilizan puede ser muy diversa, pero el fin es el mismo; representar internamente el programa para poder editar sintácticamente. Como ya se mencionó, el representar programas en forma de árboles, resulta bastante complicado, pero parece una característica inevitable. Una de las opciones fue implementada precisamente por POE, que utiliza listas doblemente ligadas para crear árboles. El EDS debe manejar algo similar, creando la representación interna por medio de listas de atributos, como en los árboles de MAGPIE, pero sin llegar a la complejidad de un árbol, quedandose exclusivamente en una lista ligada.

Ligado al manejo interno del programa, salta la descripción del manejo de expresiones, el cual es necesario hacer por separado, ya que su sintaxis no se puede checar o formatear por template. Para ello, se puede observar que todos los editores manejan expresiones como texto y después esto se verifica sintácticamente haciendolo pasar por un parser para expresiones. De este mismo modo el EDS debe manejar un analizador de expresiones para un texto tecleado libremente por el usuario.

Otro punto importante a discutir, lo representa el mane-

jo de los errores, ya que algunos editores como DICE, IPE y SAGA, solo avisan que hay error y permiten continuar la edición. Esto trae muchas dificultades si se desea hacer una edición con ventanas e incluso dificulta el manejo de la representación interna. Otros editores como MENTOR, marcan los errores en el árbol, y al final a manera de compilador, hace un recorrido que los va desplegando. Este tipo de manejo no avisa al usuario de inmediato y pierde el tiempo en un nuevo recorrido por lo que su interfase es lenta.

Finalmente se presentan en la tabla características especiales que manejan los editores, que en ocasiones son características imprescindibles en un buen programa, como son la modularización, eficiencia y rápidez. Por otro lado algunos editores muestran otro tipo de características mas especializadas que no entran dentro del alcance de nuestro editor, pero que es bueno mencionar para una futura versión. Tal es el caso de editar gráficos o manejos de edición mas sofisticados propios de un procesador de palabras o bien el proporcionar la característica del análisis semántico.

De lo anterior, se puede resaltar que se tienen características básicas con las que debe de cumplir un editor de sintaxis y que, muchas de ellas representan una misma solución compartida por los editores analizados, como es el caso del manejo de expresiones y la selección del lenguaje. Otras características presentan diversas alternativas de solución

que inicialmente aquí se presentan pero cuya decisión de implementación será tomada mas adelante, con ayuda del diseño y de las técnicas correspondientes de implementación.

IV. DISEÑO.

IV.1 Principios de diseño.

De acuerdo al Software existente, y a las tendencias del mismo, se observan ciertas características imprescindibles para el diseño del EDS de las cuales se tomaron en cuenta los siguientes principios:

- Especialización: que es la simplificación de la ejecución de tareas, incluyendo minimización de memoria y decrementar el tiempo de ejecución de ciertas tareas. Esto se logra a partir de ciertas técnicas permitidas por el lenguaje así como la interacción directa con otras caracteristicas tales como, modularidad, estructuración y optimización.

- Restricciones: se refiere a la prevención al usuario para ejecutar acciones poco productivas o dañinas. Esto se logra por la misma estructura del editor, ya que su manejo evita el mínimo de este tipo de acciones, previniendo al usuario en cada caso. Además, por efectos del diseño, se eliminará en la gramática del lenguaje, instrucciones que generen este tipo de acciones.

- Consistencia: una aproximación consistente es aquella en la que la interfase con el usuario se basa en unas pocas reglas fundamentales que habilitan al usuario a generar la mayoría de las operaciones requeridas por el sistema. Una

aproximación consistente permite al usuario aprender un nuevo sistema con facilidad sin la necesidad de aprender otros lenguajes complejos. Este concepto esta muy ligado al de los sistemas amigables que proporcionan facilidades al usuario en el uso del programa. Nuestro principal punto de enfoque para conseguir esto se basa en la sustitución de los tradicionales comandos utilizados por la mayoria de los editores, por un manejo consistente de ventanas, mas sencillo de usar y eficiente que a su vez permita el aprendizaje y conocimiento del lenguaje Pascal.

- Control manual: todos los aspectos de la programación deben estar bajo control manual, excepto cuando entran en conflicto con los principios de especialización y restricciones. Un punto a discutir aquí mas adelante es el del manejo de errores por corrección manual en vez de su corrección automatizada, lo cual esta muy ligado con lo que es la flexibilidad de un sistema, que permite a su vez operaciones estructuradas paso a paso.

- Respuesta visual inmediata: es imperativo una respuesta visual inmediata que permita al usuario monitorear el estado del sistema, corrigiendolo si es necesario. Cualquier editor orientado a pantalla debe lograr esto necesariamente. El EDS tiene el propósito de lograr ésto, con ayuda del manejo y despliegue de ventanas que permitan ver el tipo de acción que se esta realizando y también el manejo de errores que avise al usuario cuando estos se presentan, a través de

una ventana, no permitiendole continuar hasta que éstos se corrijan.

- Reversibilidad: todas las acciones deben ser fácilmente reversibles, se debe permitir al usuario el borrar y deshacer bloques que ya no sirvan, manteniéndose la sintaxis correcta del lenguaje. La reversibilidad no es posible, cuando se entra en conflicto con los tres primeros principios.

A partir de lo anterior, se trata de plantear un modelo que cumpla, dentro de los límites establecidos, los principios mencionados, éste modelo se divide básicamente en cinco módulos que son:

- Procesador de texto.
- Manejador de ventanas.
- Analisis léxico y sintáctico.
- Manejador de estructura interna.
- Manejador de comandos de edición.

Estos módulos interactúan en el EDS de acuerdo a las necesides planteadas en el análisis y se describen a detalle a continuación.

IV.2. Modelo conceptual del EDS.

A partir de los principios de diseño y por medio de una

metodología deductiva, se describe al EDS como un solo modelo donde los 5 módulos básicos interactúan entre sí, este modelo se describe en el diagrama de la figura IV.1 y se explica a continuación.

En primer lugar, el manejador del EDS lo representará el manejador de ventanas, debiendo tener el control de todo lo que sucede durante la edición así como del chequeo sintáctico es, en una palabra, la interfase con el usuario.

El manejador de ventanas entonces, llevará el control de las capas exteriores del editor, incluyendo la entrada y la salida, e interacturá con dos de los módulos básicos; el procesador de texto y el manejador de comandos. Con el primero, para aceptar la entrada de texto en el llenado de los huecos de los templates representado por identificadores, constantes y expresiones, con el segundo para permitir los movimientos en la pantalla y el programa durante la edición.

El procesador de texto además, es el enlace con el analizador sintáctico, ya que toda información tecleada por el usuario será validada sintácticamente. También, una vez validada, el procesador de texto enviará la información captada en la pantalla a guardarse en el módulo del manejo de la estructura interna.

En el caso de que la información recibida por el procesador de texto sea correcta, el control será devuelto al ma-

MODELO CONCEPTUAL DEL E.D.S.

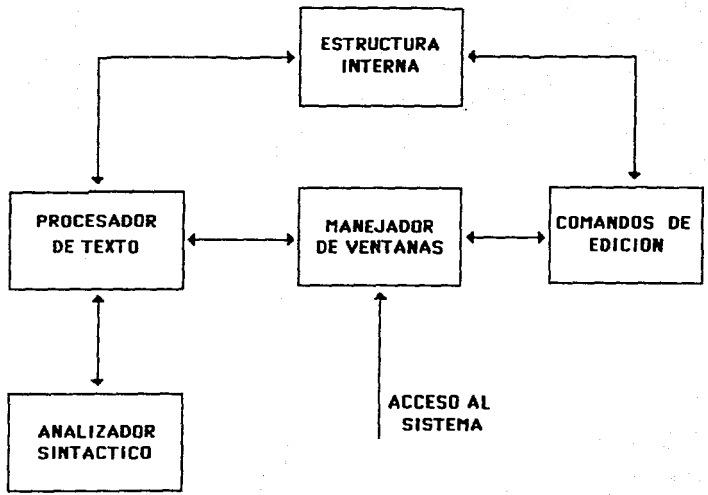


FIG. IV.1

nejador de ventanas para continuar la edición, en caso contrario, el control regresará al procesador de texto para que el usuario reintente cuantas veces sea necesario.

Por último, el procesador de comandos también podra accesar la estructura interna del programa, por medio de comandos de movimiento y copiado de bloques, e incluso por comandos de operaciones en disco.

IV.3 Diseño modular del EDS.

A partir de los principios anteriores, del análisis y contrastación de editores, y del modelo conceptual del EDS, se describen a continuación las funciones y características de los cinco módulos principales.

IV.3.1 Procesador de Texto.

Como primer paso, es necesario el tener un módulo especial que nos permita captar información del exterior, para que ésta pueda ser asimilada y procesada por el EDS. A partir de ésta característica, este módulo debe cumplir con las siguientes funciones:

. Captar la información introducida por el usuario en forma rápida y eficiente.

. Aceptar información en forma de texto pero restringiendo la entrada a un grupo de caracteres válidos.

. Proporcionar facilidades de manejo del texto como avance, retroceso, inserción y borrado de caracteres.

. Generar las palabras clave del lenguaje así como los símbolos requeridos en su construcción.

. Escribir el texto completo en la pantalla y enviarlo al módulo de la estructura interna para que se almacene físicamente.

El procesador de texto constará de tres submódulos, representados en la figura IV.2, cuyas funciones se explican a continuación.

IV.3.1.1. Captura de variables y expresiones.

Aquí se ejecuta la principal función del procesador de texto, que consiste en la captura del texto tecleado por el usuario, la cual debe estar restringida a un grupo de caracteres válidos y cuya acción será requerida ya sea por medio de las ventanas de dirección de sintaxis (en la selección de una instrucción), o bien por el analizador sintáctico (cuando lo tecleado por el usuario no sea correcto). Una vez capturada la información, la variable o expresión debe ser validada sintácticamente.

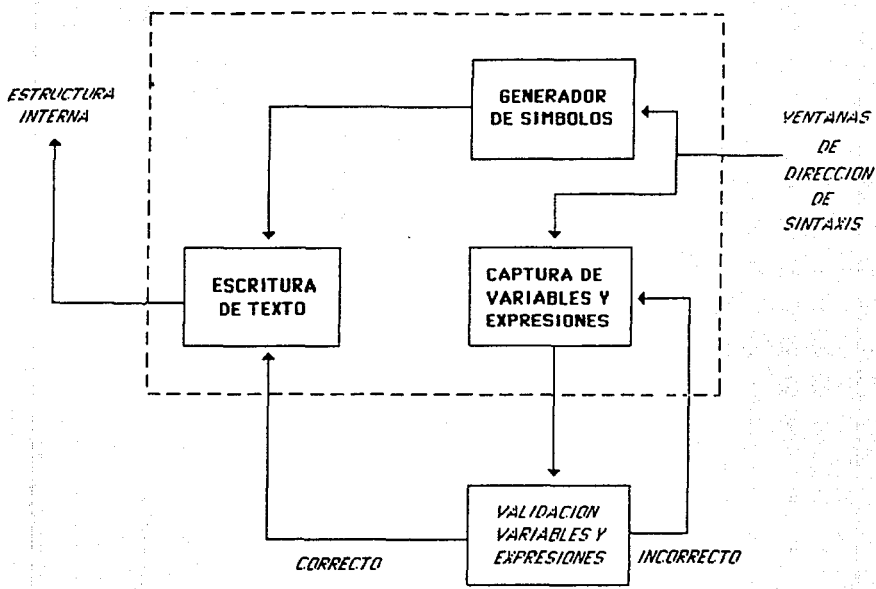

ESTRUCTURA INTERNA
GENERADOR DE SIMBOLOS
VENTANAS DE DIRECCION DE SINTAXIS
ESCRITURA DE TEXTO
CAPTURA DE VARIABLES Y EXPRESIONES
VALIDACION VARIABLES Y EXPRESIONES
CORRECTO
INCORRECTO

FIG.IV.2

IV.3.1.2. Generador de símbolos.

Este submódulo se encarga de generar las palabras reservadas y los signos de puntuación requeridos por el lenguaje Pascal, funciona bajo los requerimientos de las ventanas de dirección de sintaxis y, una vez generado el símbolo correspondiente, se pasa a escribir en pantalla.

IV.3.1.3. Escritura de texto.

Su función se limita a escribir los símbolos producidos por el Generador y el texto tecleado por el usuario ya validado y corregido. La escritura se hace sobre la pantalla y a su vez se envía al módulo de la estructura interna para que se escriba físicamente.

IV.3.2 Manejo de ventanas.

El corazón del EDS debe ser representado por éste módulo, ésto se deduce a partir de los principios de especialización y consistencia mencionados anteriormente, de tal manera que se logre una interfase con el usuario rápida, amigable y eficiente. El manejador de ventanas debe permitir entrar por menú a cualquier función relacionada con la edición, de tal manera que aquí se lleva el control de lo que entra así como de la función que se esta ejecutando. Las funciones de

éste módulo se presentan a continuación:

. Proporcionar el control de la entrada de comandos, datos, funciones e instrucciones.

. Pasar el control al procesador de texto cuando se introduzcan datos, sin perder el control global de la edición.

. Proporcionar el manejo sintáctico del lenguaje permitiendo la selección de las instrucciones por ventanas.

. Facilitar el acceso a los comandos de edición, así como a los comandos de acceso al sistema operativo y disco.

. Proporcionar información del estado de la edición.

. Avisar al usuario de las posibles acciones a tomar cuando se esta editando.

En la figura IV.3 se muestra un esquema modular del manejo de ventanas, el cual se explica a mas detalle a continuación.

IV.3.2.1. Información de ventanas

Esta parte contiene toda la información necesaria de cada una de las ventanas para poder calcular su tamaño y despliegue.

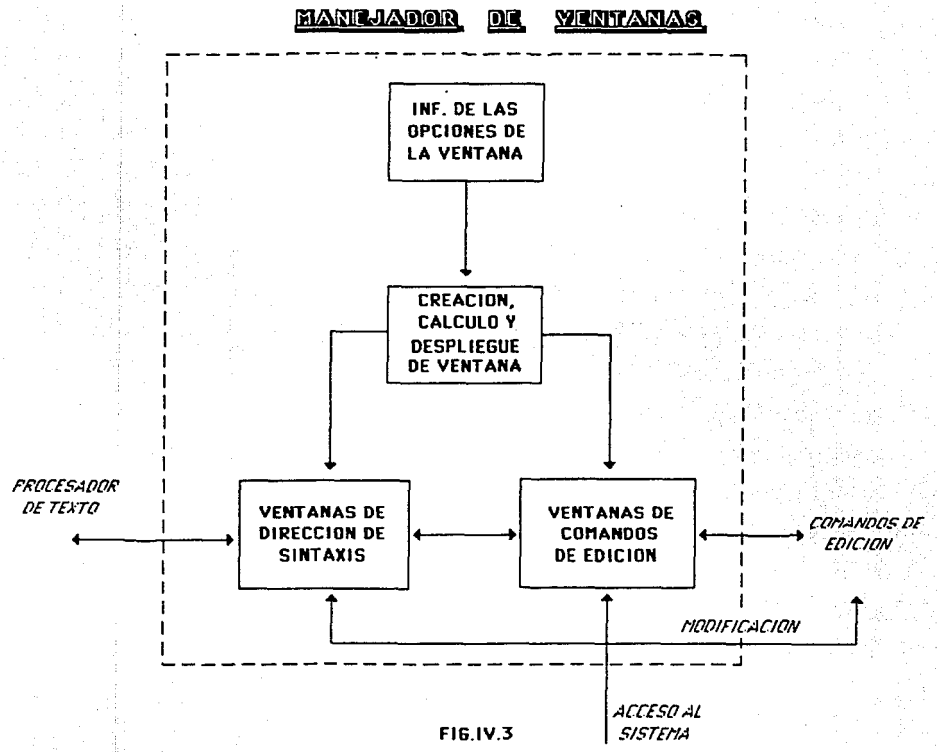

MANEJADOR DE VENTANAS
INF. DE LAS OPCIONES DE LA VENTANA
CREACION, CALCULO Y DESPLIEGUE DE VENTANA
PROCESADOR DE TEXTO
VENTANAS DE DIRECCION DE SINTAXIS
VENTANAS DE COMANDOS DE EDICION
COMANDOS DE EDICION
MODIFICACION
ACCESO AL SISTEMA

FIG.IV.3

IV.3.2.2. Creación, cálculo y despliegue.

Este submódulo se encarga de tomar la información de las ventanas, calcular sus dimensiones y desplegar la ventana con sus opciones asociadas en el lugar correspondiente de la pantalla.

IV.3.2.3. Ventanas de dirección de sintaxis.

Una vez desplegada la ventana correspondiente, el usuario podrá seleccionar las instrucciones deseadas para su programa, llevándose la dirección sintáctica a nivel ventana, es decir, dependiendo de la instrucción, se proporciona el despliegue sintácticamente correcto de las partes clave de ésta, además de que proporciona el seguimiento gramatical del lenguaje sin riesgo a violarlo.

IV.3.2.4. Ventanas de comandos de edición.

Este grupo de ventanas, es la interfase con el módulo de comandos, y su función simplemente se reduce a seleccionar comandos de ayuda a la edición como pueden ser: búsquedas, posicionamientos, llamadas a programas y directorios, etc.

IV.3.3 Analizador léxico y sintáctico.

Este módulo proporciona la característica de verificar el texto introducido libremente por el usuario léxica y sintácticamente, lo cual lo hace diferente de los editores convencionales.

Como la selección de la instrucción por ventanas proporcionará las palabras clave o palabras reservadas de la gramática de pascal, y la terminación parcial o total de la instrucción proporcionará los signos de puntuación correspondientes, el chequeo de la sintaxis sólo será requerido cuando el usuario teclee libremente texto sin restricciones (excepto el rango de caracteres permitidos por el procesador de texto) es decir, cuando teclee expresiones, además del chequeo de existencia de variables, constantes y tipos. Por lo tanto, éste módulo es una especie de "Parser" (analizador sintáctico) para expresiones y un manejador de tablas de identificadores que debe cumplir con las siguientes funciones:

. Garantizar la sintaxis correcta de lo tecleado por el usuario.

. Identificación de variables y expresiones.

. Informar al usuario de la acción a tomar cuando introduzca texto.

. Informar al usuario de los errores léxico-sintácticos a través de un submódulo con manejo de ventanas.

. No permitir el paso a las expresiones hasta que éstas

esten sintácticamente correctas avisando al usuario del error correspondiente en el momento en que sucede.

. Desglosar la información introducida por el usuario en unidades sintácticas únicas (tokens).
. Guardar todos los identificadores con los nombres de todas las variables, constantes y tipos especificados por el usuario para permitir las búsquedas correspondientes.

Un diseño detallado de éste módulo se presenta en la figura IV.4. en donde podemos observar la división de tareas en tres submódulos que interactúan de la siguiente forma para cumplir con las funciones arriba mencionadas.

IV.3.3.1. Identificador de variables y expresiones.

En esta parte, sencillamente se identifica si lo tecleado por el usuario es un identificador ó una expresión, en el primer caso la información será validada y guardada en las tablas, en el segundo, se almacenará en un área temporal para su análisis sintáctico como expresión.

IV.3.3.2. Analizador léxico.

Es necesario que las expresiones sean analizadas

ANALIZADOR SINTACTICO

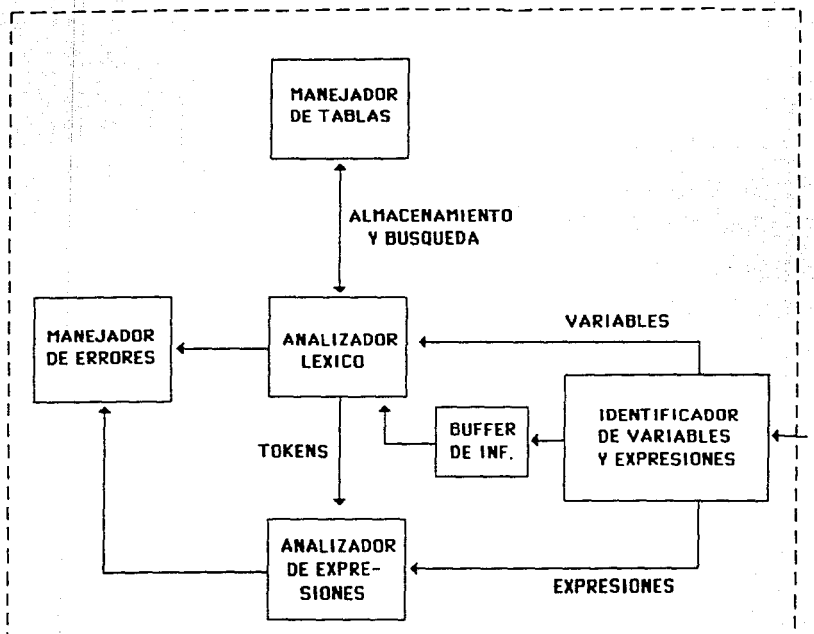


FIG.IV.4

desde el punto de vista léxico, verificándose la escritura correcta de las palabras reservadas e identificadores y caracteres válidos. Para ello el analex debe desglosar las expresiones en unidades mínimas de información denominadas "tokens" que a su vez sirven de alimentación al analizador sintáctico.

IV.3.3.3. Manejador de tablas.

Este submódulo deberá encargarse del manejo de tablas de identificadores y tipos de tal manera que sea posible un acceso tanto al guardar como al buscar información , en forma rápida y lo más eficientemente posible.

IV.3.3.4. Manejador de errores.

Este submódulo se encargará de seleccionar y desplegar en pantalla el error correspondiente a lo precisado por el analizador léxico y el analizador sintáctico. Este despliegue debe ser rápido, entendible y visible con facilidad.

IV.3.3.5. Analizador de expresiones.

El analizador de expresiones, recibirá tokens del Analex e irá siguiendo la gramática del lenguaje Pascal verificando que la sintaxis sea correcta. Cuando detecte

un error debe pasar información al manejador de errores para que que éste se despliegue inmediatamente. También interactúa con el manejador de tablas, para hacer las búsquedas necesarias de identificadores, tipos y constantes.

IV.3.4. Manejador de la estructura Interna.

La estructura interna es la forma en que se almacenará internamente el programa, de tal manera que sean posibles las funciones de edición, modificación y guardado en disco para su posterior compilación. A partir de ésto se definen las funciones principales de éste módulo:

. Almacenar en estructuras de datos el texto del programa y la información necesaria para su futura explotación

. Recibir datos del procesador de texto y almacenarlos de una manera controlada, rápida y eficiente.

. El manejo de la estructura interna debe estar asistido por el procesador de comandos de tal forma que ambos interactúen cuando el usuario desee modificar la estructura a través de comandos de inserción, movimiento y borrado.

. Debe permitir almacenar información en disco de tal manera que se pueda recuperar fácilmente, ya sea por el EDS para su modificación, o bien por un compilador de lenguaje pascal.

Básicamente, la estructura interna se guardará como una lista de nodos con toda la información requerida para poder editar, modificar y guardar en disco programas del usuario. Esta lista se irá modificando de acuerdo a 3 criterios, que basandose en la figura IV.5 se explican a continuación.

. Si se inserta información, ya sea en la creación del programa o durante su modificación, primero se crea el nodo de la lista, después se agrega su información y por último se liga a la lista en el lugar adecuado.

. Si se borra un nodo, simplemente se da de baja en la lista en el lugar escogido.

. Si se modifica la información de un nodo ya existente, primero se busca el nodo, y simplemente se sustituye su información por la nueva.

El diagrama de la figura IV.5 nos muestra lo descrito anteriormente así como la interacción de la estructura interna con los procesadores de comandos y texto.

IV.3.5. Manejador de comandos de edición.

Este módulo debe permitir la ejecución de comandos propios de un editor convencional con las debidas restricciones propiciadas por el manejo de sintaxis.

MANEJADOR DE ESTRUCTURA INTERNA

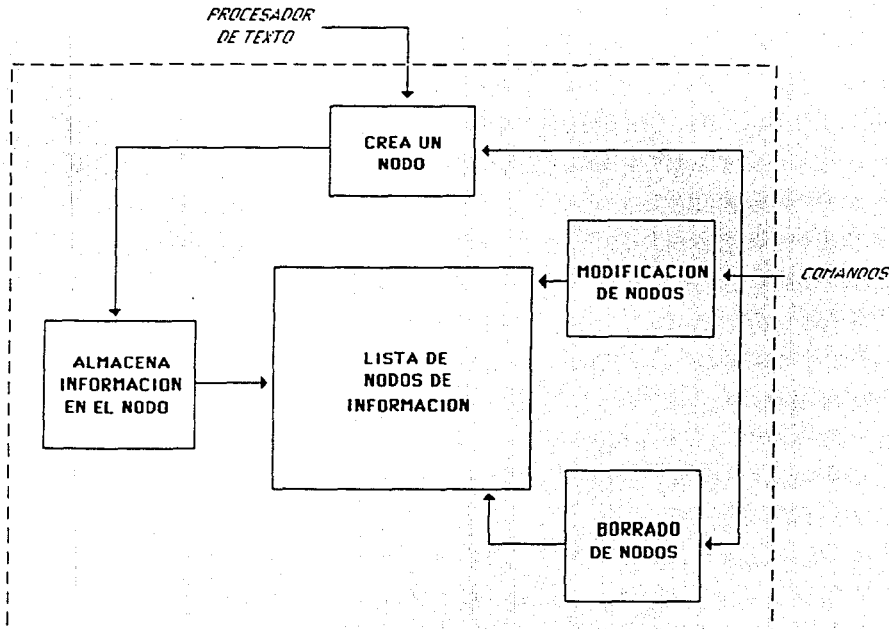


FIG.IV.5

De acuerdo a la figura IV.6 podemos observar una subdivisión de comandos en 4 grupos, independientes entre sí, y que suministran todas las funciones requeridas por un editor de sintaxis, estos submódulos se explican a continuación.

IV.3.5.1. Comandos de movimiento.

Este grupo de comandos debe permitir el movimiento del cursor dentro de la pantalla y cumplir las funciones que se enumeran continuación.

- Debe tener fácil acceso.
- Debe proporcionar los movimientos básicos sin interferir con el manejo sintáctico, tales como movimientos por página, principio y fin de archivo, de movimiento por bloque y línea.

Este tipo de comandos debe ser accesado con flechas para los movimientos básicos y por ventana para los demás.

IV.3.5.2. Comandos de modificación.

Este grupo de comandos debe permitir modificar el texto con la restricción de no destruir unidades sintácticamente correctas. Entonces, estos comandos deben permitir acciones tales como copiado y movimiento de bloques, borrado, sobreescritura e inserción, de tal forma que la sintaxis correcta del programa permanezca inalte-

MANEJADOR DE COMANDOS DE EDICION

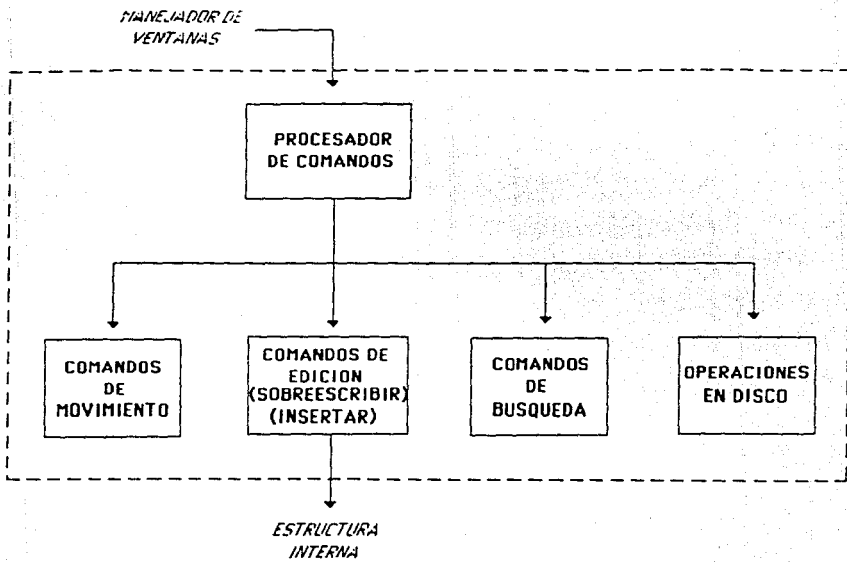


FIG.IV.6

rada y, como es lógico, ejecutándose a nivel template.

IV.3.5.3. Comandos de búsqueda.

El permitir al usuario la facilidad de hacer búsquedas, es una mejora adicional que no permiten la mayoría de los editores de sintaxis, ya que el posicionamiento del cursor debe estar restringido a lugares donde no se afecte el manejo de sintaxis. Sin embargo, es de gran ayuda en todo editor, y en ocasiones resulta imprescindible para lograr una interfase rápida, amigable y eficiente, por lo que se decide implementar este tipo de comandos controlandose su uso en la etapa de modificación.

IV.3.5.4. Operaciones en disco.

Este grupo de comandos contienen aperaciones de uso común, que facilitan la interfase con el exterior, y que son de gran importancia. Deben permitir operaciones en disco, de grabación, lectura y borrado de archivos, asi como operaciones con el directorio, cambio de nombre, etc.

V. IMPLEMENTACION.

V.1 Técnicas y herramientas utilizadas.

Para la implementación de cualquier sistema, es necesario considerar en primera instancia, las técnicas y herramientas realizadas con anterioridad para funciones afines al desarrollo, de tal forma que, aprovechandolas de una manera adecuada, se facilite y agilice la etapa de implementación, que para este caso particular representa la pesada tarea de programación. Además, éstas técnicas y herramientas proporcionan ideas de desarrollo ya probadas y adecuadas para objetivos particulares comunes de tal forma que se logra una implementación mas funcional y eficiente evitando repeticiones innecesarias en el desarrollo de nuevas técnicas.

De lo anterior, se seleccionan algunas de las técnicas que son de utilidad para desarrollar el EDS, y que de alguna manera u otra ayudaron a la implentación del mismo, o bien a la selección de la técnica o herramienta adecuada para su implementación. Estas técnicas y herramientas consultadas se presentan a continuación.

V.1.1. Técnicas para el diseño de ventanas. [19]

En cualquier sistema con desplegado de pantalla a través de mapeo de memoria, se puede implementar un sistema de ventanas de una manera muy fácil. No hay nada mágico

acerca de las ventanas, los conceptos fundamentales son fáciles de entender y usualmente fáciles de implementar. Un sistema de ventanas puede ser y ha sido implementado hasta en la computadora mas mundana.

Se asume que se necesita un grupo de rutinas de gráficos y texto, así, solo unas rutinas son esenciales; manejo de la pantalla, escritura o dibujo de una ventana, una de borrado de la ventana o de llenado, una rutina para borrar solo una porción de una pantalla donde la ventana aparecerá, una rutina que controle el posicionamiento del cursor y una para dibujo de bordes.

Una ventana es simplemente una pequeña pantalla que aparecerá dentro de una gran pantalla o sea el video, donde su función es permitirle ejecutar algunas tareas dentro de ella y entonces desaparecera cuando ya no sea necesaria. Este tipo de ventanas son conocidas como pop-up.

Así, si dos o mas ventanas son creadas, cada una con su propio propósito, se puede seleccionar la ventana a usar. Cuando una ventana es creada o abierta, esta esconde lo que se encuentre detras de ella, incluyendo las porciones de otras ventanas.

Usualmente, un borde es dibujado para delimitar la ventana y asi no crear confusiones, un titulo de la ventana es

colocado en lo alto de la misma. Teniendo abierta la ventana se pueden realizar varias funciones como: escribir texto, dibujar láminas o permitir al software realizar lo que desee.

Intentar escribir o dibujar fuera de la ventana deberá ser ignorado. Esto es permitido por medio del uso de comandos de pantalla para exportar e importar datos.

Solo se puede escribir cuando una ventana se encuentra activa, asi cuando existe un amontonamiento de ventanas, la ventana mas externa será la activa, quedando atrás las ventanas inactivas. Pero si no estan encimadas, la abierta mas recientemente será la activa. Después de cerrar una ventana, se restablecerá lo que se encontraba detrás de ella.

Un método de apertura de ventanas, es el salvar lo que se encuentra debajo de la nueva ventana y restablecerlo cuando se cierre ésta. Esto usa una gran cantidad de memoria o de espacio en disco, ya que los datos de pantalla tienen que ser almacenados en algún lugar, además de que con esto se obliga a cumplir la regla: última abierta-primera en cerrar. La ventaja de este método es que el programa no tiene que recordar como volver a dibujar lo que se encontraba anteriormente y además de que puede ser muy rápido, si es escrito apropiadamente.

La primer tarea es abrir una ventana, para eso es necesario especificar su localización en la pantalla (esquina

superior izquierda); el tamaño de la ventana (ancho y alto) y la dirección en donde se salvará lo que se encuentre debajo de ella.

Si se utiliza en el desplegado un mapeo de memoria, entonces se podrá escribir o leer de la memoria. Se entiende por mapeo de memoria, aquella área de RAM que corresponde al display, cualquier cosa que se escriba ahí se desplegará en la pantalla.

Para calcular las coordenadas en la pantalla se define la esquina superior derecha y con el ancho y largo de la ventana, se puede calcular la esquina inferior derecha a través de las siguientes ecuaciones:

x2 = (x + ancho - 1) y

y2 = (y + alto - 1)

Ahora hay que decidir la cantidad de memoria a tomar para el buffer de la ventana, una opción puede ser tomar memoria igual a la ventana mas grande que se abre en la aplicación. Pero si la memoria no es suficiente como para guardar la ventana mas grande que se abre, entonces se optaría por guardarla en disco, verificando antes si el espacio disponible es suficiente, con esta opción se pueden abrir varias ventanas y asi quedarían almacenadas en disco.

El último punto será establecer los bordes y titulos, en donde los caracteres gráficos de IBM pueden adaptarse para formar los bordes, ya que es importante enfatizar los limites de la ventana, además de que es necesario de que cuente con un título para que el usuario este enterado de las funciones que realiza esa ventana.

Cerrar una ventana es más simple que abrirla. El primer paso es recuperar la información ocupada por la ventana y para esto se necesita: localización, tamaño y la información que fue cubierta por la ventana cuando fue abierta. Como se usa la politica última abierta-primera en cerrar, la ventana que mas recientemente se abrió es la que será cerrada primero.

Usando la información de la ventana que se encuentra guardada en la RAM, se copia esta información en donde se encuentra la ventana, realizando con esto la desaparición de ésta. Si alguna ventana existe, entonces se procedería a cerrarla utilizando el mismo método hasta que todas las ventanas esten cerradas.

V.1.2. Técnicas de edición y manejo de estructura.

El editor es un programa que permite controlar un texto. Programas de este tipo se conciben para escribir o corregir el texto del que se deriva un programa (código fuente). Un editor permite desarrollar funciones tales como: escritura de

un texto, conservación y correción del mismo, anulación o inserción de partes y cambio automático de caracteres, palabras o frases completas.

El término texto se refiere a la secuencia de caracteres y/o lineas que son editadas. El texto puede ser un programa, un documento, o cualquier secuencia de uno o más caracteres. Los caracteres de una pieza de texto son usualmente representados usando el código ASCII (American Standard Code for Information Interchange). Este estándar asigna un valor numérico a cada caracter, distinguiendo, mayúsculas de minúsculas, además de incluir caracteres numéricos y algunos caracteres especiales.

Ahora, se analiza como se representa la estructura del texto de un editor. Una de las más importantes caracteristicas del diseño de un editor de texto es la estructura de datos usada para almacenar datos en memoria, mientras se esta editando. Hay varias técnicas, y cada una afecta el rendimiento de una tarea de edición especifica. Se consideran tres formas de manejo del texto, de acuerdo al incremento de su complejidad y elegancia.

V.1.2.1. Modelo de tabla de lineas.

La más simple y obvia forma de almacenar texto es utilizando un arreglo o tabla de cadena (Fig. V.1) ,

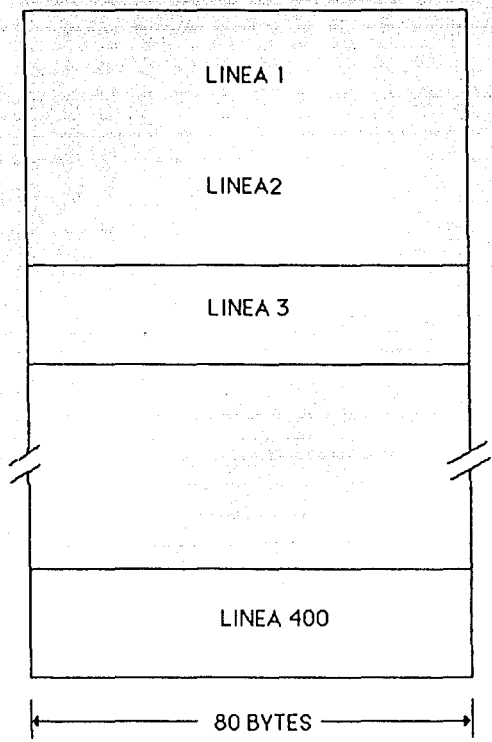


FIG. V.1 MODELO DE TABLAS DE LINEAS

cada una con un máximo de longitud. Así, si uno desea editar una pieza de texto que no tiene más de 500 lineas, y cada linea no tiene más de 80 caracteres, podemos declarar nuestro buffer como :

```
Buffer : array [1..400] of string [80];
```

Este método de fuerza bruta, mientras es extremadamente rápido y fácil, hace pobre el uso de espacio. El anexar caracteres al final de una línea es fácil, así como la búsqueda de algun caracter o palabra, sin embargo, el poner una nueva linea requiere el movimiento de un bloque, acción que consume demasiado tiempo.

V.1.2.2. Modelo de Buffer fijo.

Esta técnica también utiliza un bloque de memoria fija para almacenar texto, pero no derrocha demasiada memoria. El texto es leído en un largo y contínuo bloque de memoria. Otra ventaja es que la transferencia de texto hacia el sistema de archivos de la computadora (usualmente discos) es mas eficiente. La estructura se debilita, cuando se hace necesario introducir texto a la mitad del buffer, esto es muy lento, por la utilización del método más obvio, que es mover todo un bloque para la inserción de un caracter. Las inserciones pueden ser rápidas si se crea un espacio vacio, pero el abrir este espacio causa un gran tiempo de retardo. Este mo-

delo se presenta en la figura V.2.

V.1.2.3. Modelo de lista ligada.

En esta representación, el texto consiste de una lista ligada de registros, cada uno de los cuales contiene apuntadores a:

1) Linea anterior.
2) Siguiente linea.
3) Una cadena conteniendo una linea de texto.

La inserción de nuevas lineas es relativamente rápida y fácil. Se crea un nuevo registro dinámicamente, el cual contiene los atributos de la linea. El espacio para el texto de la linea es creado también en forma dinámica, y éste es apuntado por un campo desde el registro de atributos de linea.

La implementación es un poco más complicada que los métodos anteriores, pero ofrece mucho mayor rendimiento en todos sentidos.

Las operaciones de copiar y mover son implementadas como un simple movimiento de apuntadores. El tamaño del texto esta limitado por la memoria de la computadora, ya que ésta se utiliza dinámicamente. Este método tiene sus

BUFFER FIJO

DEFINIDO COMO UN

ARREGLO QUE PUEDE

SER MANIPULADO A

NIVEL DE CARACTER

POR EL EDITOR

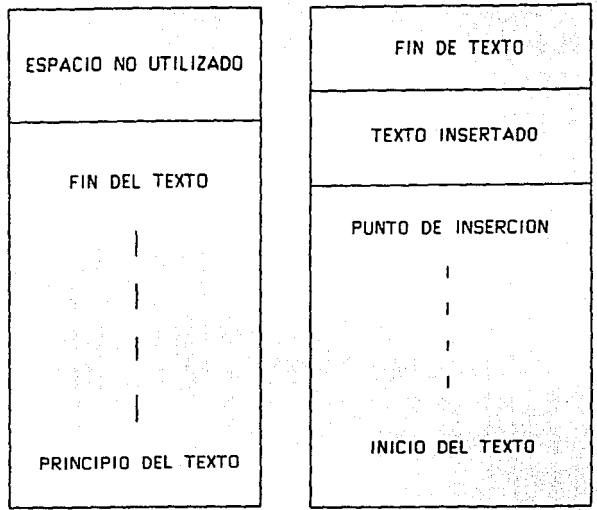


FIG. V.2 MODELO DE BUFFER FIJO

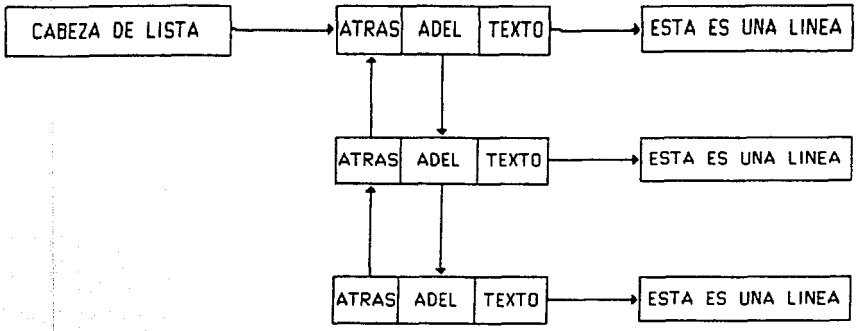
desventajas, sobre todo cuando se escribe a un archivo en disco, o se trata de leer de él, ya que cada linea debe ser creada, cada vez que se lee una nueva linea. Un esquema de éste modelo se presenta en la figura V.3.

Si el impacto de estos problemas sobre el rendimiento es minimizado, entonces el modelo de listas ligadas es superior a los métodos anteriores, así, utilizando las listas ligadas con rutinas de entrada y salida de alta velocidad, se puede obtener una alta eficiencia.

V.1.3. Técnicas de desarrollo de un analizador sintáctico.

Se pueden aplicar dos técnicas esencialmente diferentes para implementar un analizador sintáctico. La primera es diseñar un programa analizador descendente, que valga para cualquier gramática posible. En este caso se codifica una gramática especifica como una cierta estructura de datos que el programa lee, y éste opera en base a ella. Este analizador genérico esta controlado en cierta forma por la estructura de datos. Se dice entonces que el análisis sintáctico esta dirigido por tabla de estados.

La otra técnica es diseñar un programa analizador descendente que sea especifico para el lenguaje dado, y construirlo sistemáticamente según un conjunto de reglas que transforman una sintáxis dada en una secuencia de instrucciones, es decir, en un programa.



V.3 MODELO DE LISTA LIGADA

Cuando se desarrolla un analizador para un lenguaje de programación dado, apenas se necesita el alto grado de flexibilidad y parametrización que proporciona el analizador genérico y, sin embargo, un analizador especifico conduce a sistemas más eficaces y manejables, y es por ello, preferible. En ambos casos, interesa representar la sintaxis dada por un gráfico llamado gráfico sintáctico o de reconocimiento. Este gráfico representa el flujo de control durante el proceso de anáisis de una frase, éstos gráficos también son llamados gráficos de Conway.

Una característica del método descendente es que el objetivo del proceso de análisis se conoce al comienzo del mismo. El objetivo es reconocer una frase ó instrucción, es decir, una secuencia de símbolos generable a partir del símbolo inicial. La aplicación de una producción, es decir, la sustitución de un símbolo único por una secuencia de símbolos, corresponde a la división de un objetivo único en un número de objetivos parciales, a conseguir en un orden especifico. Por ello, el método descendente se conoce con el nombre de análisis por objetivos. Al construir un analizador, es fácil aprovechar ésta correspondencia obvia entre símbolos no terminales y objetivos; se construye un analizador parcial para cada símbolo no terminal. Cada analizador parcial tiene el objetivo de reconocer una subfrase generable a partir de su correspondiente simbolo no terminal. Como se desea construir un

gráfico que represente el analizador completo, se hará corresponder una subgráfica con cada símbolo no terminal.

A partir de los digramas de Conway, es fácil deducir un programa que acepte y analice sintácticamente el lenguaje Pascal. El diagrama representa, esencialmente el flujo de control del programa. Sin embargo, al desarrollar éste, es muy importante seguir un conjunto de reglas de traducción. Estas reglas se especifican a continuación, y son aplicables dentro de un marco específico. Este marco consta de un programa principal que contiene una rutina para avanzar al siguiente símbolo, y de los procedimientos correspondientes a los distintos objetivos parciales.

El programa principal esta formado por una instrucción inicial que lee el primer caracter, seguida de una instrucción que activa el procedimiento del objetivo principal del análisis. Se obtienen las rutinas individuales correspondientes a los objetivos parciales o gráficos sintácticos, aplicando las reglas siguientes. A continuación, se denomina T(S) la instrucción que se obtiene al traducir el gráfico S.

1. Por medio de sustituciones apropiadas, reducir el sistema de gráficos a un número de gráficos individuales lo más pequeño posible.

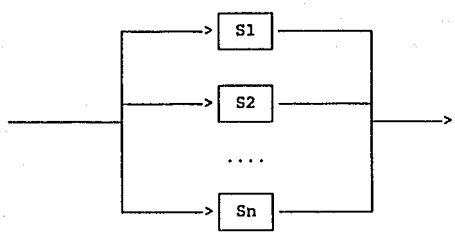
2. Traducir cada gráfico resultante en una declaración de procedimiento, segun las reglas siguientes de 3 a 7.

3. Una secuencia de elementos.

Se traduce con la instrucción compuesta:

```
begin
 T(S1);
 T(S2);
 ...
 T(Sn);
end
```

4. Una bifurcación de elementos.



Se traduce por la instrucción selectiva o condicional:

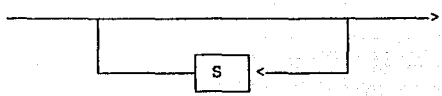
```
case ch of
L1: T(S1);          if ch in L1 then T(S1) else
L2: T(S2);          if ch in L2 then T(S2) else
...                                 ....
Ln: T(Sn);          if ch in Ln then T(S3) else
```

```
else                    error { Selección no válida }
 error;
end
```

Dónde Li designa el conjunto de símbolos iniciales de la construcción Si(Li = primero(Si)).

Si Li está formado por un único símbolo a, entonces debe expresarse « ch en Li » como, « ch=a ».

5. Un ciclo de la forma:

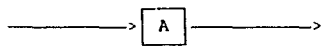


Se traduce por la instrucción:

```
While ch in L do T(S)
```

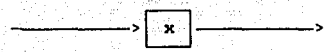
Dónde T(S) es la traducción de S según las reglas 3 a 7, y L es el conjunto L = primero(S).

6. Un elemento del gráfico que designe a otro gráfico A



Se traduce por la instrucción de llamada a procedimiento A.

7. Un elemento del gráfico que designe un símbolo terminal x.

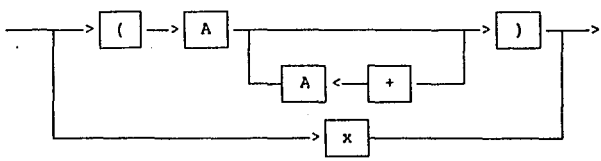


Se traduce por la instrucción:

```
if ch = x then
read(ch)
else
 error
```

Dónde error es una rutina que se activa cuando se encuentra un error sintáctico.

Se muestra la aplicación de éstas reglas traduciendo a continuación el gráfico siguiente.




```
program analizar;
var ch:char;
procedure A;
begin
if ch = 'X' then
```

```
    read(ch)
  else
    if ch = '(' then
       begin
       read(ch);
       A;
       while ch = '+' do
         begin
         read(ch);
         A;
         end;
       if ch = ')' then
         read(ch)
       else
         error;
       end
    else
      error;
  end;
  begin
  read(ch);
  A;
  end.
```

El gráfico sintáctico o de reconocimiento es una representación equivalente de la gramática del lenguaje; puede ser utilizado en lugar del conjunto de producciones en B.N.F.

(Backus Naur Form).

V.1.4. Técnica de manejo de tablas de símbolos.[20,21]

La tabla de símbolos es el medio de almacenamiento de toda la información referente a las variables y objetos en general del programa fuente que se esta editando por sintaxis.

Para poder dirigir por sintaxis es necesario disponer de la información relativa de cada elemento del programa, como es el caso de una variable o símbolo, en donde es necesario conocer :

- El tipo (entero, real, caracter, lógico, etc.).
- Si es una matriz, su número de dimensiones.
- Si es una función o procedimiento, y en su caso el número y tipo de cada uno de sus parámetros.

Por lo que es necesario agrupar toda la información sobre un elemento en una tabla de símbolos y que se podrá utilizar para verificar que un identificador o variable este declarado antes de su uso, que no se declare mas de una vez para verificar tipos de variables en una sentencia.

Es esencial que la organización y el acceso a la tabla de símbolos sea de una forma eficiente, concretamente se utiliza el doble hashing como método de acceso, porque es posible organizar la tabla de simbolos de una forma tal que con

solo un acceso (aproximadamente) se puede obtener el identificador almacenado.

El hashing consiste en determinar una función que transforme el identificador a almacenar en una dirección de almacenamiento, que esta comprendida entre 1 y m, donde m es el tamaño de la tabla, de forma tal, que el número de colisiones sea pequeño o nulo. Se denomina colisión al fenómeno de que dos claves diferentes den la misma dirección propia.

Formalmente una función de hashing, es definida como un mapeo :

H : K --> A

Donde:

H, es una función de hashing.

K, variable o llave de la función.

A, dirección obtenida al aplicar la función H a la llave K.

Una buena función de hashing debe cumplir con dos requerimientos:

a) Su ejecución debe ser muy rápida.

b) Esta debe minimizar el número de colisiones.

La primera es algo que depende de la máquina y la segunda depende de los datos. Muchos métodos han sido sugeridos,

pero ninguno de estos ha superado el método de división, la cual se define como:

$$h(x) = (x \bmod m)+1$$

En donde:

x, es la llave en turno.

m, es el tamaño de la tabla a manejar.

El mejor resultado del método de la división se obtiene cuando la tabla de tamaño m es primo (es decir , m no es divisible por ningún entero positivo diferente a 1 y m), con lo que se tienen menos colisiones. Este método tiene el problema, de tener efectos de amontonamiento o agrupamiento debido a las colisiones que se tienen, haciendo la búsqueda lenta. Sin embargo este fenómeno de amontonamiento o (clustering), puede ser aliviado por un doble hashing; en esta técnica el valor de la dirección es encontrado usando una segunda función de hashing h2(k), la cual es independiente de la función de hashing inicial h1(k).

El doble hashing utiliza dos funciones de hash, h1(k) y h2(k). Usualmente h1(k) produce valores entre 0 y m-1 inclusive, pero h2(k) debe producir un valor entre 1 y m-1 que es relativamente primo a m. Su algoritmo es el siguiente ya sea para búsqueda o inserción:

```
D1,[Primer hash].        i <-- h1(k)
D2,[Primera prueba].     Si TABLA[i] = vacía entonces ir a D6
```

```
                        de otro modo si LLAVE=k entonces FIN.
D3,[Segundo hash].      c <-- h2(k)
D4,[Avanza al prox.].   i <-- i-c
                        Si i <0 entonces i <-- i+m
D5,[Compara]            Si TABLA[i] =  vacía entonces ir a D6
                        de otro modo
                        Si LLAVE = k entonces FIN.
                        de otro modo regresar a D4
D6,[Insercion]          si n  =  m-1 entonces existe overflow
                        de otra forma
                        n <-- n+1, TABLA[i] = ocupada,
                        LLAVE <-- k.
```

Donde LLAVE es el nodo que contiene la llave y posiblemente otros campos. Una variable auxiliar n es usada para guardar la cantidad de nodos ocupados y se incrementa en uno cada vez que un nodo es insertado.

Hay varias posibilidades para ejecutar h2(k); si m es primo y hi(k) = k mod m, se puede asignar

```
h2(k)= 1+(k mod (m-1));
```

pero si m-1 es par, es mejor asignar

```
h2(k)= 1+(k mod (m-2));
```

esto sugiere escoger m tal que m-1 y m-2 son dos numeros primos como 319 y 317.

Varios programadores, tienen gran fé en algoritmos y

ellos se sorprenden al encontrar que la forma obvia de suprimir registros de una tabla no es un trabajo fácil.

En general, se puede manejar supresiones de la tabla, poniendo códigos especiales en la celda correspondiente. Así, hay 3 clases de entradas a la tabla; vacío, ocupado y suprimido. Cuando se busque una llave debe saltarse aquellas celdas marcadas con suprimido, como si estas estuvierán ocupadas. Si la búsqueda no es exitosa, la llave puede ser insertada en el lugar de la primera supresión o en la posición vacia encontrada.

Pero esta idea de trabajar solo cuando hay supresiones es muy rara, por que la entrada de la tabla nunca estara vacia de nuevo, una vez que estas han sido ocupadas.

. Después de una larga secuencia de repetidas inserciones y supresiones, todo el espacio vacio eventualmente desaparecerá, y cada búsqueda no exitósa tomara m pruebas, por lo que el tiempo por prueba se incrementara.

El algoritmo para la supresión es el siguiente:

R1.[Vacía una celda] TABLA[i] <-- vacio, j <-- i

R2.[Decrementar i] i <-- i-1, Si i<0 entonces i <-- i+m

R3.[Analiza TABLA[i]] Si TABLA[i] = vacia entonces Fin.
de otro modo r <-- h(LLAVE).
La dirección original del hash de la

llave es almacenada en la posición i.
Si (i <= r < j) o si (r < j < i) o
Si (j < i <= r), es decir si r
se mantiene ciclicamente entre i y j,
ir a R2.

R4.[Mover un reg.] TABLA[j] <--- TABLA[i], ir a R1.

V.1.5. Detección,reporte y corrección de errores.

La aparición de errores durante la creación de un programa es muy frecuente, pero desgraciadamente, su detección solo se logra hasta el proceso de compilación.

El editor dirigido por sintaxis, no trata de verificar la lógica del programa, ni la posiblilidad de error en iteraciones infinitas que tampoco detecta un compilador normal, se trata pues, de detectar y recuperar en la medida de lo posible las propias anomalías susceptibles de ser observadas bajo criterios léxicos y sintácticos.

La presencia de un error sintáctico se detecta cuando en el proceso de análisis, no han sido halladas las producciones o reducciones respectivamente necesarias para el reconocimiento de la cadena de tokens, pero esto se evita y se recupera en forma inmediata, ya que la dirección por sintaxis, lleva en forma intrínseca la acción de evitar los erro-

res sintácticos.

Una vez que un error ha sido detectado, debe de ser reportado al usuario, con su diagnóstico asociado. Se entiende por diagnóstico de error a aquella cadena de caracteres que emite el editor con el fin de notificar al usuario de la presencia de un error o contingencia ocurridos durante el propio proceso de edición. Los diagnósticos están por tanto, lo más directamente relacionados con el error que pretenden denotar.

Algunos programas utilizan las rutinas estándar del lenguaje para imprimir un pequeño mensaje, también es común imprimir un código de error en vez de mensajes naturales, que son descritos a detalle en un manual, teniéndose el inconveniente de tener que contar con el manual en forma permanente. La generación de códigos de error en lugar de mensajes completos no es muy justificada excepto en pequeños microcomputadores.

Una aproximación a los mensajes de error en lenguaje natural, se logra generando y salvando códigos de error en un archivo temporal y después en una pasada final se imprimen sustituyendo el código por el mensaje correspondiente. Esta aproximación puede prescindir del manual de errores y probablemente es el estándar mínimo aceptable para reportar errores.

Para elaborar mensajes efectivos, se utiliza la aproximación de manejador de mensajes, cuyo objetivo es minimizar el espacio mientras maximiza la flexibilidad y consistencia de los mensajes de error. La manera de implementar un manejador de mensajes es por medio de una tabla que contenga el número de error asociado y su mensaje correspondiente. El procedimiento se reduce a suministrar el número de error al manejador, que lo utilizará como índice de la tabla para desplegar el mensaje.

V.1.6. Nombres de comandos para editores de texto(ET). [22]

Los problemas de los usuarios se ven reflejados en la cantidad de errores que cometen. Puede parecer obvio que un usuario latino cometa mas errores al usar programas cuyos mensajes y vocabularios estan en inglés. Pero si asi fuera, ¿qué tanto depende del usuario y que tanto de la máquina?, ¿qué procedimientos podemos diseñar para disminuir las tasas de errores y acelerar el aprendizaje?.

Por otro lado, es conveniente tener criterios de evaluación conductual de los editores que puedan ser aplicados por el diseñador o el comprador nacional, pensando en el usuario hispano-parlante. tales criterios permitirian comparar diferentes paquetes de programas (sistemas operativos, editores de texto, paquetes de análisis estadísticos, etc.) con base en la eficiencia con que pueden ser usados.

Miller y Thomas (1977) [23] encontraron que el 75% de los comandos activados por un sistema IBM fueron comandos de edición. En términos de la frecurencia de uso, un editor de textos es la mas importante facilidad proporcionada por un sistema de cómputo. Si el editor de una computadora está inadecuadamente diseñado, el sistema completo puede ser rechazado por los usuarios.

Al manejar un editor, el usuario debe de identificar la tarea a realizar y recordar el nombre de la misma. Por ejemplo si el usuario quiere borrar una linea debe recordar que, "borrar" es el nombre del comando. Además el usuario tiene que recordar el formato con que se identifica la parte del texto a modificar. (Van Dam & Rice 1971 [24]).

Desde el punto de vista psicológico, se dice que un conjunto de nombres para un editor es óptimo, si los nombres se aprenden, recuerdan y se utilizan más facilmente que cualquier otro conjunto de nombres. (Scapin, 1981 [25]).

Para que un conjunto de nombres sea óptimo, tales nombres debieran de estar en el lenguaje natural de los usuarios. Ledgar y colaboradores (1980) [26], mostraron que un editor cuyo vocabulario estaba compuesto por palabras familiares, descriptivas (semánticamente relacionadas), era mas fácil de aprender y usar que otros sistemas con la misma potencia operacional pero con diferente estructura sintáctica. Los sujetos realizaron mas adecuadamente las tareas de edi-

ción con el primer editor. Los autores concluyen que al rediseñar la sintáxis superficial del editor comercial utilizado, haciendolo mas parecido al lenguaje natural de los usuarios, se produjo una mejoría significativa en la eficiencia de uso del sistema.

El diseño adecuado de los nombres de los comandos, asi como las especificaciones de formato deberían verse reflejadas en una mayor eficiencia de uso del sistema. Para ello se requieren procedimientos de evaluación conductual de los sistemas. Roberts & Moran (1982,1983) [17], han identificado 4 dimensiones básicas de evaluación:

1) Tiempo para realizar tareas estándares de edición.
2) Cantidad de errores cometidos al realizarlas.
3) Tasa de aprendizaje de las tareas básicas de edición.
4) Porcentaje de tareas preespecificadas.

Al usar un ET el usuario tiene que recordar el nombre del comando que se aplica a la operación que se pretende realizar. Parece intuitivamente claro que algunos nombres de comandos sean mas fáciles de recordar que otros.

Rosenberg (1982) [18], sugiere que la bondad del nombre de un comando puede estimarse considerando la relación que los usuarios hacen de los nombres con las acciones que denotan, es decir, un conjunto de nombres de comandos es bueno

en la medida que:

1) Sugiere lo que hace el comando.
2) Sugiere la relación (similar, opuesto, etc.) de ese comando con los otros del sistema.

Para que el nombre sugiera la acción, debe existir una correspondencia entre el conjunto de rasgos de los nombres y de las acciones, por lo que un nombre es bueno si su significado (conjunto de rasgos) es similar al significado del comando.

En conclusión, las variables importantes a considerar para dar nombres a las acciones de un ET son:

1) La frecuencia del nombre.
2) La discriminabilidad de nombre en el conjunto.
3) La similitud semántica entre el nombre y la acción denotada.

V.2. Defininición del medio ambiente inicial de trabajo.

Para poder iniciar la implementación del EDS, es necesario definir primero el medio ambiente inicial de trabajo. En base a la evaluación del análisis y la contrastación de editores de sintaxis, se toman ciertas decisiones de diseño y se generan ciertos criterios iniciales para la implementación

que proporcionan un primer medio ambiente susceptible a modificarse dependiendo de la evaluación y de la programación. Estos criterios que nos definen el medio ambiente inicial de trabajo son:

- El EDS debe asegurar una sintaxis correcta del programa editado pero sin tratar con la semántica ni información dependiente de contexto.

- El EDS debe proveer todas las palabras clave necesarias, así como separadores, terminales y todo lo sintácticamente requerido por el lenguaje.

- El EDS servirá única y exclusivamente para lenguaje Pascal debido a su gramática estructurada y su masiva implementación.

- El EDS se basará en el seguimiento estricto de la gramática de Pascal para controlar la sintaxis. En caso de que alguna estructura gramatical entre en conflicto con la implementación, podrá ser eliminada.

- El EDS utilizará templates ya que es la estructura ideal para el manejo sintáctico, ya que su tiempo de creación es rápido y minimiza errores de tecleo al eliminar tareas mundanas de edición.

- La interfase con el usuario se dará a través de un manejo de ventanas por medio del cual deberán ejecutarse todos los comandos así como las instrucciones en la edición.

- La representación interna del programa se realizará por medio de listas de atributos doblemente ligados debido a su implementación rápida y sencilla.

- El manejo de expresiones debe ser inmediato, tomando la información como texto y analizandola sintácticamente, de manera que no se permita continuar la edición hasta que la expresión sea correcta.

- La modificación del programa solo será factible a nivel template o bien a nivel expresión. No debe permitirse cambiar palabras reservadas ni signos de puntuación. El EDS solo podrá modificar programas creados por el EDS.

- Solo es permitida la inserción y el borrado a nivel template.

V.3. Descripción del lenguaje a implementar, Pascal.

Un algoritmo o programa de computadora consiste de dos partes esenciales, una descripción de acciones, las cuales seran ejecutadas y una descripción de los datos, los cuales seran manipulados por estas acciones. Las acciones son descritas por las instrucciones, y los datos por las declaraciones y definiciones.

Cada programa consiste de un encabezado y un bloque. El bloque contiene una parte de declaración, en la cual todos los objetos locales al programa son definidos; y una parte de instrucciones, que especifican las acciones a ser ejecutadas sobre estos objetos.

El encabezado proporciona un nombre al programa y una lista de sus parámetros, que con las nuevas versiones del

lenguaje para microcomputadoras, ya no son necesarios, como por ejemplo la definición de archivos de entrada y de salida, ya que la comunicación del programa con el ambiente ya esta predefinida.

El bloque consiste de seis secciones donde cualquier sección, excepto la última puede ser vacía. Sin importar el orden de las cinco primeras, estas secciones son:

Declaración de etiquetas.
Declaración de constantes.
Definición de tipos.
Declaración de variables.
Declaración de funciones y procedimientos.
Instrucciones.

La primera sección lista todas las etiquetas definidas en el bloque. Las instrucciones de los programas en lenguaje Pascal resultantes, son ejecutadas en la secuencia en la cual aparecen en el texto del programa, sujetas a los patrones de control, impuestos por las diferentes instrucciones estructuradas. Estos patrones de control son siempre suficientes para expresar las acciones requeridas de un programa, por lo que es innecesario el utilizar etiquetas y la instruccion goto, por razones de eficiencia y sobre todo de claridad del programa, ya que pueden desviarse del orden implícito de ejecución asociado con estas instrucciones.

La consistencia estructural de un programa expresado en términos de las construcciones if...then...else, case...of..., while...do..., repeat....until..., etc, esta garantizada por las reglas mismas del lenguaje. La consistencia del mismo programa expresado con instrucciones goto es responsabilidad del programador.

La segunda sección define sinónimos de constantes, por ejemplo, se pueden introducir identificadores que mas tarde pueden ser usados, en lugar de una constante en general.

El uso de identificadores constantes hace que un programa sea mas fácilmente leible y ayuda a crear una documentación conveniente. Además, permite cambiar sus valores de una manera fácil, ya que no es necesario realizarlo uno a uno, creando la portabilidad y modularidad del programa.

La tercera sección consiste en la definición de tipos. El tipo de dato define el grupo de valores que una variable puede tomar. Cada variable que aparece en un programa debe ser asociada con un solo tipo. Existen en Pascal los tipos escalares, los cuales son predefinidos y son: integer, real, byte, boolean y char.

Sin embargo se tiene la posibilidad de que el programador defina sus propios tipos. La definición determina un grupo de valores y asocia un identificador con este grupo de valores.

La declaración de variables debe preceder textualmente a cualquier utilización de la variable. La declaración asocia un identificador y un tipo de dato con una nueva variable, simplemente listando el identificador requerido por su tipo.

La asociación identificador-tipo es válida en todos los aspectos para el bloque, que contiene la declaración a menos que el identificador este redefinido en un bloque subordinado. Esto es aplicable en los bloques anidados.

Los procedimientos y funciones deben estar definidos antes de su uso. Los procedimientos son subrutinas y son activadas por instrucciones de procedimiento. Las funciones son subrutinas que producen un valor resultante, y por lo tanto pueden formar parte de expresiones.

Finalmente aparece la sección de instrucciones, que indica como se van a manipular los datos definidos anteriormente. Las instrucciones son simples, como por ejemplo una asignación o llamada de procedimiento; o también pueden ser estructuradas y compuestas como las instrucciones repeat...until..., while...do..., etc. La instrucción fundamental es la asignación y especifica que un nuevo valor calculado sera asignado a una variable a través del operador asignación :=. El nuevo valor se obtiene por la evaluación

de una expresión consistente de variables, operadores y funciones.

Una expresión es una construcción que denota reglas de ejecución para obtener valores de variables y generar nuevos valores por la aplicación de operadores. Las reglas de composición especifican precedencia de operadores de acuerdo a cuatro clases de operadores. El operador not tiene la más alta precedencia, seguido por los operadores de multiplicación, después los operadores de suma y finalmente con mas baja precedencia estan los operadores relacionales. Secuencia de operadores con la misma precedencia son ejecutados de izquierda a derecha.

Las instrucciones compuestas especifican que las instrucciones son ejecutadas en la misma secuencia que ellas son escritas. Los símbolos begin y end actúan como instrucciones delimitadoras de bloque. Note que el cuerpo de un programa tiene la forma de una instrucción compuesta. Pascal utiliza el punto y coma (;) para separar instrucciones.

Instrucciones de repetición especifican que ciertas instrucciones deben ser ejecutadas en forma iterativa, dependiendo de la condición establecida. Entre estas se encuentran: while...do..., repeat...until... for...to/downto..

Instrucciones de condición especifican que una instrucción debe ser ejecutada solo si una cierta condición es verdadera de otra forma se ejecutará la siguiente instrucción al término del bloque. La instrucción case consiste de una expresión o seleccionador y una lista de instrucciones, cada una de las cuales deben estar etiquetadas por una constante del tipo del selector.

V.4. Variaciones con respecto al lenguaje Pascal estándar.

Debido a las limitantes naturales de una edición de sintaxis controlada por ventanas (entre otras), es necesario restringir la gramática del lenguaje Pascal a las estructuras representadas en el apéndice A por diagramas de Conway. Estas estructuras son casi la totalidad de las propuestas por N. Wirth [3] en la definición del lenguaje, e incluso con algunas mejoras. Por lo tanto, se enumeran a continuación las principales variantes del lenguaje Pascal utilizado por el EDS y el del Pascal estándar.

- La sección de declaración de etiquetas desaparece, atendiendo la necesidad de eliminar la instrucción goto cuya libertad de uso resulta peligrosa en una edición de sintaxis ya que puede destruir estructuras e influir en el programador para que pierda el poder intrínseco del lenguaje estructurado.

- La longitud de las cadenas para identificadores queda

restringido al de uso estándar que es de 8. Las palabras reservadas utilizadas son básicamente las mismas que las de pascal estándar y se presentan en el apéndice C.

- El uso de funciones y procedimientos de librería, se restringe a los proporcionados en la tabla IV del apéndice C, que se basan en los utilizados por el Turbopascal comercial versión 3.1, que superan con mucho los del lenguaje estándar. En las tablas de funciones se pueden incluir funciones de los paquetes Toolbox de Borland para lenguaje Pascal.

- El tamaño de las instrucciones queda limitado al tamaño de la línea de edición que, como máximo es de 256. Pero solo se declara una instrucción por línea, para poder tener un máximo control con el formateo del programa, ya que éste es indentado por el EDS en forma automática.

- Se declara una variable por renglón, para que el manejo de ventana por parte del usuario resulte más efectivo. Esto también se aplica con la declaración de constantes y tipos.

- No es necesario declarar en el encabezado del programa, los archivos de entrada y salida que se utilizan durante la ejecución de éste, ya que el lenguaje Turbopascal los maneja en forma intrínseca.

V.5. Selección y justificación del lenguaje de implementación.

Este aspecto debe cubrirse antes de la codificación y después de la etapa de análisis. Al seleccionar el lenguaje de programación, el factor más importante es minimizar el costo de la vida de la programación, tanto en desarrollo, como en mantenimiento.

Esta selección debe hacerse en las siguientes 3 etapas:

a) Inicial: Análisis no cuantitativo, se analiza la compatibilidad entre el lenguaje y procesador, y lenguaje y aplicación.

b) Intermedia : Se emplean criterios cuantificables, tales como tiempo de ejecución, similitud entre lenguajes, versatilidad, etc.

c) Análisis Formal : Se basa en criterios administrativos y técnicos:

   i) Criterios Administrativos: tiempo y costo de desarrollo y mantenimiento.

   ii) Criterios técnicos:

      ii.1) De primer orden: Representación de datos, estructuras de control

y programación de sistemas.

ii.2) De segundo orden: Transportabilidad, facilidad de aprendizaje, documentación, eficiencia en espacio, inteligibilidad, etc.

La versión de lenguaje Pascal de la compañia Borland International (TurboPascal V.3.01) que corre en las computadoras personales y compatibles, presenta las siguientes caracteristicas comunes a todos los demás lenguajes de alto nivel:

Estructuras de control.

```
if condición then condición else ...
While condición Do ...
Repeat ... Until condición
For ... to/downto ... do
Case variable of
etc.
```

Manejo de archivos.

```
Secuencial
De acceso directo
```

Además de tales caracteristicas, cuenta con las siguientes, particularmente ventajosas:

- Creación y control de nodos, en forma dinámica.
- Manejo de interrupciones y funciones del sistema opera-

tivo, para control del software y hardware.

- Control absoluto del video y teclado, a través del direccionamiento directo a memoria, esto con ayuda de interrupciones y funciones.
- Utilización de archivos de otros programas (Including Files) y Overlays.
- Manejo de recursión.
- Posibilidad de incluir programas en ensamblador.

La transportabilidad esta garantizada ya que genera un archivo COM o EXE (versión 4.0); es un lenguaje de fácil aprendizaje, ya que es estructurado; y por ésta misma razón es inteligible; genera programas que crecen poco en espacio, además de rápidos en se ejecución.

De este modo se opto por realizar el EDS en lenguaje TurboPascal, que garantiza un desarrollo y operación eficaz del EDS.

V.6. Implementación del Procesador de texto.

El funcionamiento de este módulo es requerido por el módulo manejador de ventanas, el cual le indica si va a generar un símbolo o palabra reservada o va a capturar una variable o expresión. Para el caso de la generación de símbolos o palabras reservadas se realizan de acuerdo al template requerido por el manejador de ventanas pasándolo a escribir

a la pantalla y subsecuentemente al módulo manejador de la estructura interna. Si el requerimiento es la captura de una variable o expresión el procesador de texto cuenta con un procedimiento llamado PIDECADENA, el cual se encarga de capturar caracter por caracter la información tecleada por el usuario. Una vez terminada la captura, la cadena es validada en el módulo de análisis sintáctico el cual definirá si es correcta o no. En el caso de algún error se deberá de capturar nuevamente la cadena hasta que ésta sea correcta, cuando ésto se cumpla la cadena se sobreescribe en pantalla cambiando su intensidad, enviando el requerimiento para el manejo de la estructura interna.

Las rutinas que componen éste módulo se describen a continuación.

```
Pidecadena (Renglón : Integer    { Posición horizontal }
            Columna : Integer    { Posición vertical   }
            Longitud : Integer   { Tamaño máximo       }
            Carlegal : Carválido { Caracteres válidos  }
        Var Cadena : Cadenas     { Variable a editar   }
        Var Tecla : Byte );      { Tecla terminal      }
```

Este procedimiento se encarga de capturar la Cadena tecleada por el usuario caracter por caracter dentro del conjunto de caracteres válidos de Pascal. Es posible proporcionarle la Longitud de la cadena a capturar de tal manera que

se puede controlar el tamaño de las variables (8 caracteres) o expresiones (256 caracteres). La tecla terminal durante la captura de información, esta representada por el caracter ASCII número 13 que corresponde al retorno de carro. También se permite la edición completa de la Cadena, ya que cuenta con los movimientos a los lados con flechas, facilidad de inserción (cambiando al cursor cuadrado) y supresión de caracteres. Adicionalmente cuenta con la facilidad de controlar el posicionamiento en la pantalla, actualizando simultáneamente las coordenadas indicadas por el editor en la parte superior derecha de la pantalla.

Manda_Cadena (Cadena : string [256]); Este procedimiento se encarga de manipular la variable o expresión una vez ya validada y correcta, actualizando el estado de las coordenadas de la pantalla y mandando al procedimiento Procesa_Texto caracter por caracter.

Procesa_Texto (Ch : Byte); El procedimiento recibe un caracter de Manda_Cadena y la escribe en pantalla en la posición indicada por sus apuntadores, a su vez verifica el fin de línea para realizar el scroll y colocando el caracter en el campo Linea_Texto que forma parte de la estructura de datos del EDS. En caso de no haber memoria disponible se genra un error indicando la falla al usuario.

La generación de símbolos es una función controlada por el manejador de ventanas y no representa un procedimien-

en especial sino que actúa cuando es requerido en base al template seleccionado.

V.7. Implementación del Manejador de Ventanas.

Este módulo se encarga de controlar todas las funciones relacionadas con la edición y dirección sintáctica durante la creación o modificación de un programa, representando la interfase con el usuario.

Este manejador se basa en un sistema de manú Pulldown, cuya implementación se realizó utilizando rutinas de ventana tipo Pop-Up mencionadas en las técnicas de implementación cuya función genérica es salvar la información sobre la cual se va a sobreescribir y la restablece una vez que se cumple una determinada función.

El menú Pulldown se construye utilizando una estructura de datos sobre la cual se definen cada una de las opciones del menú proncipal. Cada una de éstas opciones cuenta con su propia estructura de datos dónde se encuentran los submenús asociados a cada opción.

La rutina de inicialización del menú toma la estructura de datos y en base a ella, crea, calcula y despliega el menú principal. Después de la inicialización y en base a la opción seleccionada del menú principal, se ejecuta una rutina

de inicialización de submenús encargada de crear, clacular y desplegar el submenú correspondiente. Una vez desplegado el submenú se captura la opción seleccionada, ejecutándose la rutina correspondiénte y pasándo el control al procesador de texto cuándo se esta editndo o bien al procesador de comandos cuando lo requiera.

V.7.1. Información de las opciones de las ventanas.

Como ya se mencionó, la información de las ventanas es una estructura de datos implementada en base a un arreglo secuencial de direcciones absolutas que permiten un mejor control sobre los datos, en el que se incluyen solamente el título de la opción, su longitud y el caracter que va a permitir su selección directa, la declaración de ésta información esta presentada en el apéndice C.

V.7.2. Creación, cálculo y control.

El tamaño de la ventana a desplegar es calculado por el procedimiento Inicia_Menu en base al título con mayor longitud del submenú. Una vez calculado el tamaño se procede a desplegar en pantalla el contorno y el contenido del submenú, función realizada por el procedimiento Dibuja_Borde.

V.7.3. Ventanas de dirección de sintaxis.

La dirección de sintaxis es realizada por los procedi-

mientos Programa, Bloque y Declaración. Cada procedimiento realiza su función en base a una ventana representada con el mismo nombre. La dirección de sintaxis se inicia en el procedimiento Programa el cual genera la primera línea del programa y pasa el control al procedimietno Bloque que a su vez lo pasa a Declaración, una vez que se ejecute su función. Cada vez que se hace una elección sobre la ventana correspondiente, el editor toma la elección y la compara con las definidas por el EDS. Lo anterior esta implementado con la instrucción Case-of y de esta manera cualquier instrucción del lenguaje Pascal que se analize, es desglosada para poder aplicar la producción adecuada de la gramática de Pascal. Cada producción tiene tanto elementos terminales como no terminales. Cuando se encuentra un terminal el procedimiento en ejecución termina, en caso de ser un no terminal se procede a invocar el procedimiento correspondiente hasta que se llegue a un elemento terminal. En el caso de que una instrucción contenga un error sintáctico, este será detectado ya que siempre, indistintamente se tiene que finalizar con un elemento terminal cumpliendo estrictamente con la gramática del lenguaje.

Observando la figura V.4., se muestra que si el flujo de control es seguido rigurosamente se evitará cualquier error de sintaxis. Como se puede notar, la instrucción por sí misma indica el flujo de control del programa, así por ejemplo en el procedimiento Bloque, si se elige la opción

FLUJO DE VENTANAS DE SINTAXIS

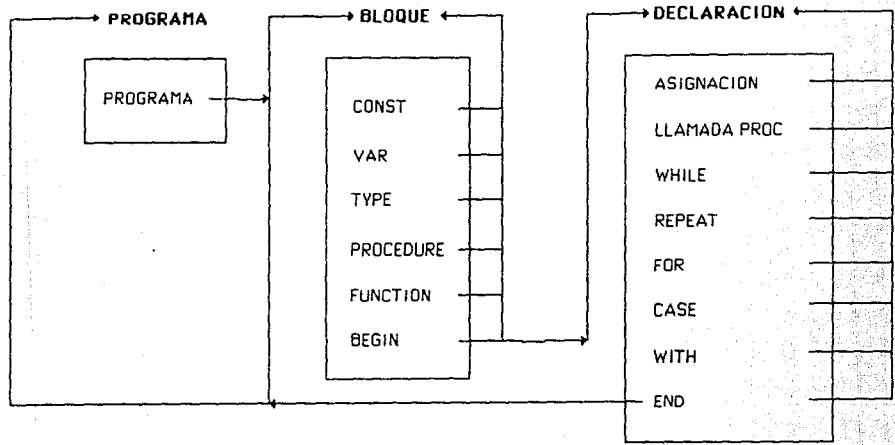


FIG. V.4

Const, al término de su ejecución el control regresará invariablemente al procedimiento Bloque, no siendo así, si se elige el delimitador Begin que invoca al procedimiento Declaración, éste toma entonces el control de flujo del EDS.

Los procedimientos Bloque y Declaración se encuentran siempre en una iteración hasta que sea seleccionada una acción de entrada a otro Bloque o de finalización a través de los delimitadores Begin y End respectivamente.

El anidamiento de procedimientos, funciones e instrucciones se logra fácilmente utilizando la recursividad, es así que podemos tener un Procedure dentro de otro Procedure, invocando de nuevo el procedimiento Bloque desde el Bloque mismo, la sintaxis por lo tanto sigue siendo controlada por el procedimiento Bloque pero a otro nivel de recursión. Con ésto se obtiene que el código de control de sintaxis sea corto, no redundante y con la facilidad de anidar instrucciones a cualquier nivel de recursión.

Con el método anterior se logra el control de sintaxis y recupera en forma automática los errores ya que las palabras reservadas son controladas por el EDS, así como los delimitadores, conjuntando finalmente el template y evitando errores de estructura. La función del usuario entonces, consiste en introducir las variables y expresiones que desee manejar. Tanto las variables, como las expresiones son validadas por un analizador léxico, un analizador de expresiones

y un manejador de tablas, todos éstos interactuando con un manejador de errores (Módulo analizador sintáctico).

V.7.4. Ventanas de comandos de edición.

Es el conjunto de ventanas del menú principal que controlan la selección de los diferentes comandos que interactúan en el medio ambiente de un editor. La facilidad de cada ventana es proporcionarle al usuario la lista de los comandos posibles a ser utilizados para que sean ejecutados por el módulo correspondiénte. Las ventanas de comandos de edición se muestran en la figura V.5.

Este módulo se compone de las siguientes rutinas:

BorraSubMenu(Col,Ancho:Integer); Borra el menú que actualmente se encuentra desplegado en pantalla y restablece el texto original que se encontraba en pantalla, conociendo la posición del menú sobre la pantalla, además de el ancho.

EscribeMenuPrin; Presenta en la parte superior de la pantalla las diferentes opciones con que cuenta el EDS, como son Archivo, Buscar, Posicionar, Bloque, Editar y Modificar.

EscribeSubMenu(Opción,Col,Ancho:Integer); Presenta la ventana con las diferentes opciones del submenú, éste es escogido de acuerdo a la variable opción, que va de 1 a 6; se

VENTANAS DE EDICION

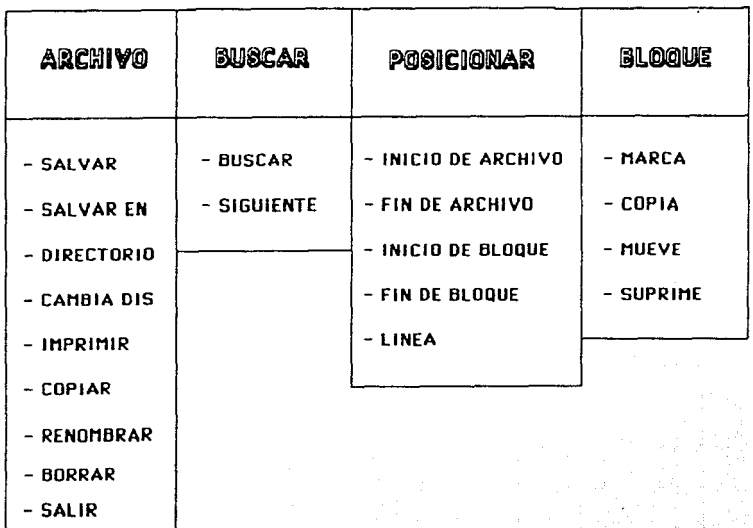


FIG. V.5

proporciona el tamaño de la ventana con los parámetros Col y Ancho, que indican posición en la pantalla y ancho de la ventana.

IniMenuPrin; Inicializa las rutinas principales, variables, estructuras de datos; relacionados con el menú y manejo de ventanas.

IniSubEstr(Opción:Integer); Inicializa la estructura de datos, de acuerdo al submenú elegido.

ControlaMenu; Esta rutina controla todo el sistema de menús con que cuenta el EDS. Primero inicializa completamente al sistema manejador de menús y ventanas. Espera a una elección del usuario y determina que submenú será desplegado. Después de que es desplegado el submenú la rutina vuelve a esperar una elección sobre el submenú y ejecutar la acción deseada mandando llamar la rutina adecuada.

HazVentana(Col,Lin,Ancho,Largo:Byte); Crea una ventana en el lugar especificado por las variables Col,Lin,Ancho y Largo. Se utilizó la técnica de manejo de ventanas presentada en el primer punto de éste capítulo.

RestableceVentana(Col,Lin,Ancho,Largo:Byte); Borra la ventana y restablece la información que se encontraba hasta antes de crearla.

SalvaPantalla; Salva la información en donde se va a crear una ventana y esta información es almacenada en un arreglo para que posteriormente sea reestablecida.

DibujaBorde; Presenta en pantalla la ventana y el borde pero no la información, ésta es proporcionada por el procedimiento EscribeMenuPrin y EscribeSubMenu utilizando la pequeña estructura de datos que fue inicializada.

V.8. Implementación del analizador sintáctico.

De acuerdo al diseño especificado en el capítulo anterior, se definieron 5 bloques para realizar el análisis léxico y sintáctico de expresiones. A partir de estos bloques y atendiendo a las especificaciones de diseño, se describe a continuación la implementación de cada uno de ellos.

V.8.1. Identificación de variables y expresiones.

Para identificar que lo tecleado por el usuario es un identificador o una expresión, no se implementa un módulo en particular, ya que la misma dirección de sintaxis global dada por el manejo de ventanas nos proporciona la ubicación del usuario a nivel Bloque o Declaración, siendo para el primer caso una expresión y para el segundo un identificador (tipo,constante o variable).

V.8.2. Analizador léxico.

El analizador léxico es un Scanner implementado en forma genérica, es decir, con tabla de estados, donde la tabla de estados esta representada internamente por un arreglo fijo que corresponde a la estructura de datos que controla al analizador.

Para implementar un analizador genérico, es necesario en primer lugar, obtener los autómatas finitos para el conjunto de tokens de la gramática en cuestión, para el caso del EDS se define una gramática restringida mostrada en el apéndice A. En cuanto a los tokens, estos se definen en la tabla I del apéndice C, donde cabe aclarar la necesidad de utilizar como tokens específicos a el número real y entero, así como a la constante de caracteres para facilitar su reconocimiento en lugar del token número y del apóstrofo.

A partir de éstos tokens se elaboran los autómatas para cada uno de ellos de manera que se genera una secuencia de estados que nos servirá para su seguimiento en la tabla de estados, éstos autómatas se muestran en el apéndice B.

Una vez terminados los autómatas, se forma la tabla de estados utilizando como columnas los caracteres permitidos por el lenguaje y como renglones a los estados de los autómatas, la intersección en la tabla será el estado siguiente del proceso de análisis léxico de tal forma que si es un nú-

mero positivo se trata de un estado terminal, si es un número negativo, se trata de un estado no terminal al que esta asociado un token único, finalmente si el estado no tiene intersección (cero), el estado terminal anterior nos representa el token.

Al presentar la tabla de estados varios renglones vacíos, se optimiza el espacio haciéndola más pequeña unificando todos los renglones vacios en un solo estado de tal manera que se optimiza el tamaño y el tiempo de acceso de la tabla. Ambas tablas se presentan en el apéndice C.

Adicionalmente el scanner es el responsable de buscar si el token encontrado es una palabra reservada o no, y en caso de serlo asignarle el número de token que le corresponde. También es el responsable de detectar caracteres inválidos, aunque la mayoría de ellos son filtrados por el procesador de texto.

Las rutinas que forman el analizador léxico se explican a continuación:

```
Scanner (Var Token:Integer;        { Número de token }
         Var Cade:String[8];       { Identificador   }
         Var Cadena:String[256]);  { Expresión       }
```

El scanner es el procedimiento principal, y consta de

tres parámetros pasados por valor, Token es simplemente el número de token encontrado, Cade representa el identificador o símbolo que corresponde al token encontrado y Cadena es la expresión tecleada por el usuario que saldrá sin la parte correspondiente al token que se encuentre después de la llamada. Por ejemplo en la expresión de entrada: UNO < DOS , Cadena = "< DOS", Cade = "UNO" y Token = 1 al salir de la llamada.

Getchar (Var C:String[1]): Procedimiento que desglosa la cadena de la expresión caracter por caracter.

Getnum (Var C:String[1];Var N:Integer): Procedimiento que obtiene la coordenada N de la columna de la tabla de estados a partir del caracter C.

Borra: Procedimiento para borrar los blancos de la cadena de la expresión.

Busca: Búsqueda binaria de la cadena del token encontrado (Cade), en la tabla de palabras reservadas. La posición encontrada sumada a 25 (que representan los primeros tokens, ver tabla I apéndice C) nos proporciona el número de token de la palabra reservada, en caso de no encontrarlo se asigna el token 1 que representa a los identificadores.

Analex (Var Token:Integer): Procedimiento iterativo que controla al scanner y asigna el número de token correspon-

diente.

V.8.3. Manejador de tablas.

El manejador de tablas se implementa en base a una función llamada internamente como Hash, la cual se encarga de guardar y buscar los identificadores en la tabla de símbolos, utilizando el método de hashdoble. Al ser el hash doble la técnica de manejo de tablas más rápida y eficiente fué seleccionada en esta fase de implementación. La tabla consta además, de todas las especificaciones del identificador útiles para su reconocimiento, y que son explotadas por los demás módulos en su oportunidad, éstos datos adicionales se presentan a continuación en la esturctura de datos de Pascal, para la tabla de identificadores.

```
Identificador = Record
        Nombre : String[8];
        Clase  : Char;
        Tipo   : Byte;
        Param  : Byte;
        Nivel  : Byte;

Tabla_Id = Array [320] of Identificador;
```

Donde:

. Nombre representa el nombre del identificador.

. Clase se refiere a si es constante, variable, etc. Para el EDS este campo esta dado como sigue:

P : Procedure F : Function
C : Constante V : Variable
T : Tipo J : Conjunto
C : Subrango Y : Arreglo
A : Apuntador L : Archivo
R : Registro S : Set
I : Identificador de programa

. Tipo es el tipo escalar definido por el pascal estándar, o bien tipos definidos por el usuario. Los tipos definidos en el EDS son:

1 : Integer
2 : Real
3 : Byte
4 : Char
5 : String
6 : Boolean
7 : Otros (Conjuntos, Subrangos, Apuntadores, Arreglos, Archivos, Registros)

. Param es el número de parámetros que contiene el identificador en el caso de ser un procedimiento o función.

. Nivel es el nivel de anidamiento de la variable cuando es declarada (siendo 0 para las globales).

Además, la tabla de identificadores contiene también los procedimientos y funciones dados por el lenguaje Pascal que se presentan en la tabla IV del apéndice C.

Las rutinas que forman parte del manejador de tablas son :

Function HashDoble: Función que controla todo el manejo de la tabla de identificadores.

Procedure Hashing: Procedimiento recursivo que obtiene la llave del identificador y envía requerimientos de error al manejador de errores en casos como sobreflujo, repetición de un identificador, identificador ilegal ó ya declarado, etc.

Asigna_Llave: Asigna las características del identificador en la llave correspondiente.

Function Hash: Ejecuta la función de hash en base al método de división. Utiliza como módulo 317 para el primer hash, y 319 para el segundo.

Procedure Hashbusca (Id:String[8]): Procedimiento de búsqueda de identificadores, que utiliza las mismas funciones para encontrar el identificador Id deseado.

V.8.4. Manejador de errores.

Para el desplegado del diagnóstico, se utiliza la técnica de generador de mensajes, en donde el procedimiento ERRORES, recibe un número de error, ya sea del SCANNER o del ANAEXP, y accesa el mensaje de una tabla almacenada en disco, de tal manera que su implementación resulta muy sencilla y su eficiencia muy alta ya que se ahorra tiempo de proceso y espacio en memoria. Además, el mensaje desplegado se envía a la parte central de la pantalla, de tal forma que es inmediaatamente visible por el usuario, para desaparecer, basta oprimir cualquier tecla. La lista completa de los errores detectados por el EDS se muestran en la tabla V del apéndice C.

V.8.5. Analizador de expresiones.

El analizador de expresiones es un pequeño parser ó analizador sintáctico, exclusivo para expresiones de Pascal. Esta implementado con un analizador descendente particular basado en la gramática presentada en los diagramas de Conway del apéndice A.

El Anaexp recibe del exterior una cadena con la expre-

sión correspondiente la cual va siendo desglosada en tokens por el Scanner cuando se requiere, cada token debe corresponder a un token válido dentro de la secuencia sintáctica englobada por las reglas gramáticales de Pascal mostradas en los diagramas de Conway del apéndice A en la parte que involucra Expresion.

El procedimiento es recursivo y se termina hasta terminar con la expresión, si se detecta un error durante el proceso, el usuario debe de corregir la expresión para volver ha ser analizada hasta que este completamente correcta.

El anaexp es el responsable de llamar al procedimiento Hash cuando el token identificador es enviado por el Scanner de tal forma que el Hash busca el identificador reportando de su existencia o no.

El Anaexp tiene como rutinas principales:

Anaexp (Cadena : String[255]): Boolean : Es la función que engloba todo el analizador de expresiones tomando el valor de verdadero cuando la expresión es válida. Cadena es la expresión a analizar.

Expresion; Parte del analizador que llama al procedimiento Expresion_Simple, siendo recursivo cuando detecta los símbolos de comparación (=,>,<,>=,<=,<>,IN).

Expresion_Simple; Procedimiento de verificación de expresiones simples que incluyen las operaciones con +,- y OR , llamándo a Term en su siguiente fase.

Term; Procedimiento de verificación de expresiones no simples que incluyen los símbolos * y / , y las palabras reservadas MOD y AND llamándo en su siguiente fase a Factor.

Factor; Caza el token enviado para determinar una cierta acción que incluye la posibilidad de volverse a llamar recursivamente cuando encuentra la palabra reservada NOT, o bien al regresar de la llamada de TERM. Además identifica las constantes, variables y Funciones, haciéndo la busqueda correspondiente en la tabla de identificadores. Puede contener nuevas expresiones al detectar el paréntesis cuadrado o redondo, o bien dentro de una función.

Variable; Localiza la variable e identifica su posición, detecta variables de tipo apuntador al encontrar el token 24 (^), variables tipo registro con el token 6 (.) o variables tipo arreglo con el token 10 ([) para el cual llama al procedimiento Tipoa.

Tipoar; Procedimiento para validación de rangos, es llamado por Factor al detectar el token 10 ([), llama a validar una expresión y después la existencia del token 21 (..) para establecer un rango, el token 7 (,) llamandose así

mismo o bien el token 11 (|) para terminar.

Tipos; Procedimiento de validación de parámetros. Cuando se trata de una función valida que haya sido declarada y acepta una expresión a validar, si ésta es correcta el procedimiento ya sabe cuántos parámetros debe contener. Finalmente sólo valida los parámetros como expresiones y las comas con el paréntesis que cierra.

Tipoa; Procedimiento de validación de arreglos, internamente llama a expresión y después valida que contenga el token 7 (,) volviéndose a llamar, o bien el token 11 (|) que cierra la validación.

Busca_Campo; Búsqueda especial en el árbol de campos.

V.9. Implementación del Manejador de la Estructura Interna.

Este módulo se encarga de crear la configuración y manipular la estructura interna de los datos. Su funcionamiento es requerido tanto por el procesador de texto como por el procesador de comandos.

La representación interna del texto es una lista ligada de lineas, cada una con un apuntador hacia adelante, uno hacia atrás y un apuntador al fin o inicio de template, un contador de longitud de linea y un apuntador a una cadena

creada en forma dinámica, además de un caracter indicador de tipo de instrucción y una variable utilizada como bandera que muestra si la linea forma parte de un bloque marcado. La representación en pascal del registro manejado es el siguiente:

```
type
     Linea_Texto  = string[255];
     ApDesc_Linea = ^ Desc_Linea;
     ApLineaTexto = ^ Linea_texto;
     Desc_Linea   = record
                         Liga_Ade : ApDesc_Linea;
                         Liga_Ante: ApDesc_Linea;
                         Liga_Temp: ApDesc_Linea;
                         Txt      : ApLineaTexto;
                         Bandera  : integer;
                         Log_Linea: integer;
                         Tipo_Inst: char;
                    end;
```

La representación gráfica se presenta en la figura V.6.

. El procesador de texto es el encargado de hacer el requerimiento de creación de nodos, cuando es necesario crear una linea de texto conteniendo una instrucción. Una vez creado el nodo el procesador de texto envia la información de la instrucción a ser almacenada en la estructura interna. Finalmente se incluye esta linea a la lista de líneas ligada actualizando apuntadores de liga_ade y liga_ant. La liga

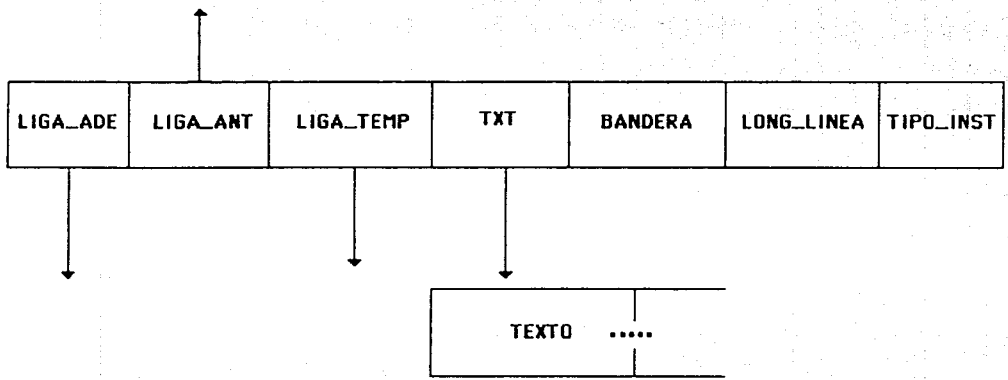


FIG. V.6 ESTRUCTURA INTERNA DE LINEA DE TEXTO

de inicio o fin de template (liga_temp) es actualizada por el módulo analizador sintáctico. Esta actualización se lleva a cabo en base a un apuntador generado en cada nivel de selección de la instrucción de tal forma que cuando la instrucción ha sido completada la parte final apunta a la inicial y viceversa, como es el caso de la linea Repeat que apunta a la línea Until. Lo anterior se especifica con la figura V.7. En este momento se asocia un caracter con el tipo de linea de programa, estos caracteres son utilizados posteriormente en la fase de reconstrucción de ligas, función necesaria cuando se desea modificar un programa ya cre ado, asi como para los comandos de edición Marca, Copia, Mueve y Suprime bloque. Los caracteres asociados a cada instrucción para su reconocimiento se presentan a continuación.

A : Asignación	B : Begin
C : Case-of	D : Record
E : End	F : For
H : With	I : If-Then
J : Procedure	K : Function
L : Else	O : Const
P : Program	Q : Repeat
T : Etiqueta (Case)	U : Until
W : While-do	Y : Type
V : Var	

Las rutinas principales que forma el módulo manejador

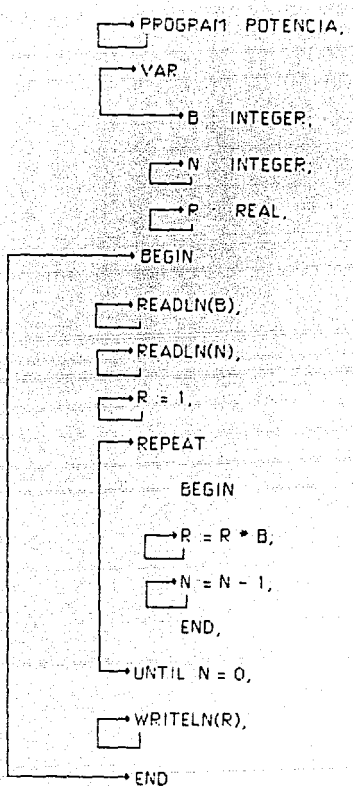


FIG. V.7 LIGA DE INICIO O FIN DE TEMPLATE.

de estructura interna son:

CreaNodoTxt; Esta rutina crea dinámicamente un registro Desc_Línea y LíneaTexto en forma conjunta. El contenido de estos nodos es limpiado o inicializado.

BorraNodoTxt; Borra dinámicamente un registro Desc_Línea y su registro asociado LíneaTexto.

TamañoLinea(p:ApDesc_Línea; Ncols:integer); Examina el espacio que tiene disponible en el buffer de la línea para el nodo a donde apunta p; esto se logra consultando el campo log_línea que contiene el tamaño de la línea en cuestión , y si es más pequeño que Ncols, una nueva línea es almacenada con espacio suficiente y la línea anterior es liberada de memoria. Controla la cantidad de memoria disponible en la computadora y si no hay suficiente espacio, manda un error indicandolo.

SuprimeLinea(p:ApDesc_Línea); Esta rutina borra una línea del texto y se asegura de no modificar la estructura interna de los datos.

InsertaLínea; Crea una nueva línea, apoyandose en las rutinas CreaNodoTxt y TamañoLínea, la línea puede ser insertada al principio, a la mitad o al final del texto. Maneja en forma completa los apuntadores liga_ade, liga_ante y liga_temp.

V.10. Implementación del manejador de comandos.

La implementación de éste modulo se llevó a cabo utilizando rutinas independientes que realizan las funciones especificadas en el diseño.

El manejador de ventanas envia el requerimiento del comando seleccionado por el usuario a través de la elección de una opción en la ventana, entrando a este modulo por medio del procesador de comandos que es el encargado de identificar el tipo de operación a realizar la cual puede ser: de movimiento, de edición (sobreescribir e insertar), de búsqueda o de disco.

V.10.1. Comandos de movimiento.

Los movimientos permitidos sobre el programa editado son:

- Línea por línea, hacia arriba y hacia abajo. (Utilizando la tecla con la flecha correspondiente).
- Por página. (Teclas PgUp y PgDn, para página arriba y página abajo, respectivamente).
- Posicionamiento al inicio o fin de archivo.
- Posicionamiento al fin o inicio de un bloque marcado.
- Posicionamiento directo a una línea en particular.

El movimiento línea por línea y los movimientos por página se realizan solamente cuando se esta a nivel menú prin-

cipal de otra manera, las teclas dedicadas a estos movimientos tienen una función diferente, como por ejemplo, en una ventana de submenú, ya que las flechas seleccionan directamente una opción de éste.

La implementación de estos movimientos es muy sencilla, porque para subir una línea basta con cambiar el apuntador a la Liga_Ante y si el movimiento es a la línea siguiente el apuntador cambiara a la Liga_Ade de la estructura interna. Para el caso de los movimientos de página el desplazamiento del apuntador es en bloques de 20 líneas.

El posicionamiento al inicio o fin de archivo se implementa, también en forma sencilla, ya que se tienen los apuntadores al inicio y fin del programa, basta entonces posicionarnos en estos apuntadores para lograr el movimiento deseado.

Para implementar el posicionamiento a fin o inicio de un bloque marcado se cuenta con dos apuntadores llamados Inicio_Bloque y Fin_Bloque, los cuales apuntan a las líneas marcadas una vez que se ha utilizado con anterioridad la opción Marcar.

Para posicionarse en una línea directamente, aparece una ventana en la que se pide al usuario introducir el número de línea en la que se quiere posicionar, una vez tomado este da-

to el editor se posiciona al principio del archivo y apartir de ahí comienza a contar las líneas hasta llegar a la seleccionada.

Las rutinas principales que componen esta serie de comandos son las siguientes:

Inc_Línea; Esta rutina incrementa el número de línea y la despliega sobre la ventana de estatus.

Dec_Línea; Rutina que decrementa el número de línea para posteriormente actualizar la ventana de estatus.

Pág_Ade; Rutina que incrementa el apuntador 20 líneas para actualizar la ventana de estatus.

Pág_Atrás; Esta rutina posiciona al apuntador 20 líneas atrás, actualizando la ventana de estatus.

Inicio_Archivo; El apuntador se actualiza con el valor del apuntador de inicio de archivo y se actualiza la ventana de estatus.

Fin_Archivo; Posiciona al apuntador al fin del programa que se edita, igualando su valor al del apuntador de fin de archivo.

Comienza_Bloque; Coloca al apuntador al inicio del blo-

que que se ha marcado, si no existe un bloque marcado, el usuario recibe un mensaje indicándoselo.

Termina_Bloque; Actualiza al apuntador con el valor de Fin_Bloque, actualiza la ventana de estatus.

Ve_a_Línea(Núm_línea:integer); Esta rutina posiciona el cursor en Núm_Línea. Si Núm_Línea es mayor que el número de lineas con que cuenta el programa, el cursor se coloca en la última linea del archivo.

V.10.2. Comandos de Edición.

Este tipo de comandos se utilizan para modificar el texto, ya sea por medio de copiar, mover o suprimir bloques, o bien utilizando la opción de inserción y sobreescritura para crear nuevas instrucciones para el programa.

Para poder insertar, se utilizan los mismos procedimientos de la etapa de creación del programa con la opción Editar (Programa, Bloque y Declaración) con la diferencia de que éstos no actúan en forma conjunta, sino en forma independiente. Estos procedimientos se activan dependiendo del lugar en donde se piensa insertar. Si nos encontramos en una declaración de variables, la ventana que se activa es la de Bloque, y si nos encontramos en la parte de declaración de instrucciones del programa principal o de un procedimiento se ejecutará la

rutina de Declaración, teniendo en cuenta de que no se pierde el manejo de recursividad e identación.

Para sobreescribir, primero es necesario posicionarse en la línea sobre la que se sobreescribirá y aparece una ventana que pide la palabra a cambiar y la cadena por la cual será sustituida. La nueva cadena es también validada sintácticamente, lo cual implica que la sobreescritura es válida solo para expresiones completas o nombres de variables existentes.

Las principales rutinas que componen este tipo de comandos son:

Marca_Bloque; Rutina para marcar un bloque en donde el apuntador Inicio_Bloque es actualizado con el valor de la línea en donde se desea empezar a marcar. Como las estructuras a marcar solo pueden ser templates, Fin_Bloque toma el valor de la Liga_Temp que es el apuntador al final del template.

Copia_Bloque; Procesa el comando sobre el bloque, copiando las lineas de texto en el rango del bloque definido por los apuntadores Inicio_Bloque y Fin_Bloque, en el lugar donde esta posicionado el cursor.

Mueve_Bloque; Realiza la misma función que la rutina Copia_Bloque, pero no permite que el movimiento sea dentro del bloque y se suprime el bloque de su posición original.

Suprime_Bloque; Borra las líneas que se encuentran entre los apuntadores Inicio_Bloque y Fin_Bloque. Todas las líneas son borradas con el procedimiento Suprime_Línea del manejador de la estructura interna.

V.10.3. Comandos de búsqueda.

Este tipo de comandos nos permiten realizar búsquedas de cualquier cadena dentro del texto del programa, una vez encontrada, el cursor se posiciona al inicio de la línea en donde se encuentra ésta. La búsqueda da opción de pasar a las siguientes ocurrencias de la cadena.

La principal rutina que interviene en la búsqueda es:

Busca_Palabra; Procedimiento de búsqueda de una cadena a partir de la línea en donde esta posicionado el cursor, hasta la primera ocurrencia de ésta.

V.10.4. Operaciones en disco.

Este tipo de comandos nos permiten comunicarnos con el sistema operativo de la computadora y así contar con la facilidad de salvar y recuperar archivos, imprimirlos, copiarlos, renombrarlos o borrarlos, además de tener la posibilidad de listar el directorio del disco o cambiar de manejador de disco, sin la necesidad de abandonar el EDS.

Las rutinas principales que componen estos comandos son:

Salva_Archivo; Almacena todo el texto en un archivo en disco. Esto se realiza utilizando las funciones de MS-DOS, $3C, $40 y $3E que sirven para crear, escribir y cerrar un archivo, respectivamente.

Salvar_En; Se apoya en la rutina anterior con la diferencia de que solicita al usuario el nombre del nuevo archivo con que se va a salvar en el disco.

Directorio; Rutina que obtiene el directorio especificado por el usuario, utiliza las funciones de MS-DOS,$4E y $4F.

Cambia_Disco; Cambia a un nuevo manejador de disco y se implementa con la instrucción ChDir del lenguaje TurboPascal.

Imprimir; Rutina de impresión basada en el barrido de la estructura interna, línea por línea, mandando respectivamente a la impresora, pero llevando el control del número de líneas impresas por hoja, además de incluir un encabezado.

Copia_Archivo; Copia un archivo utilizando los procedimientos de TurboPascal BlockRead y BlockWrite y apoyándose en la rutina de Salva_Archivo.

Renombrar; Rutina que renombra un archivo y esto se lo-

gra utilizando el procedimiento Rename, implementado en TurboPascal.

Borrar; Rutina para borrar archivos, que utiliza el procedimiento de TurboPascal Erase.

VI. EVALUACION DE RESULTADOS.

Después de probar con diversos programadores de diferentes niveles de experiencia, fue posible evaluar de una manera más objetiva el editor dirigido por sintaxis (EDS). Para ello hay que considerar que actualmente no existe un programador que domine el editor y que los programadores escogidos fueron indistintamente expertos o novatos del lenguaje Pascal, e inclusive programadores que lo desconocen totalmente.

VI.1. Ejemplo prototipo.

A continuación se mostrará un ejemplo de la edición de un programa sencillo, escrito en lenguaje Pascal con el fin de evaluar y ejemplificar el funcionamiento del EDS. El programa evalúa la función trigonométrica coseno utilizando la expansión : $\cos(x) = 1 - x^2 / (2*1) + x^4 / (4*3*2*1) - \ldots$
es el siguiente:

```
Program Coseno;
Const
     eps = 1E-14;
Var
     x : Real;
     sx : Real;
     s : Real;
     t : Real;
```

```
        i : Integer;
        k : Integer;
        n : Integer;
    Begin
    Read(n);
    For i:= 1 to n do
      Begin
        Read(x);
        t := 1;
        K := 0;
        s := 1;
        sx := sqr(x);
        While abs(t) > eps * abs(s) do
          Begin
            k := k + 2;
            t := -t * sx / (k * (k - 1));
            s := s + t;
          End;
        writeln(x,s,k div 2);
      End;
    End.
```

Al iniciar la sesión, el EDS preguntará por el nombre del programa, así como si el programa es nuevo o ya existe, tal como lo presenta la figura VI.1 que se presenta a continuación.

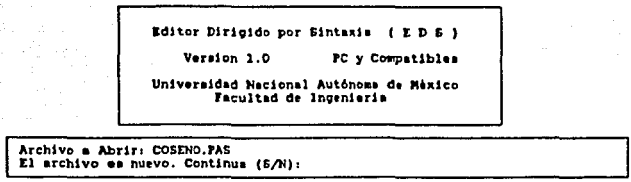


Figura VI.1

Una vez pasado la pantalla de presentación, lo primero que aparece es el menú principal del EDS que, para editar contiene una ventana propia. Para iniciar la creación del programa es necesario posicionarse en la palabra Program. Figura VI.2.

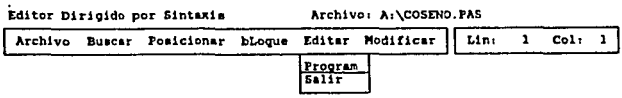


Figura VI.2.

Al seleccionar la palabra Program, el editor la genera para escribirla en pantalla, colocando el cursor inmediatamente después esperando por un identificador para darle nombre al programa. En la parte superior izquierda de la pantalla se indica al programador lo que esta esperando el editor, en este caso un identificador. Lo anterior es mostrado en la figura VI.3.

Figura VI.3.

· Una vez tecleado el nombre del programa, aparece el submenú de declaraciones, en el que escogeremos la palabra Const para introducir la segunda línea del programa. Los signos de puntuación " = " y " ; " son generados automáticamente por el editor. Figura VI.4.

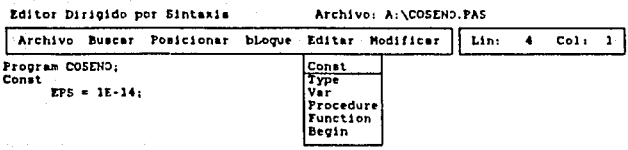


Figura VI.4.

Al terminar de escribir nuestra constante, el control es devuelto a la ventana de declaraciones, en donde se selecciona ahora la palabra Var para introducir nuestras variables. Figura VI.5.

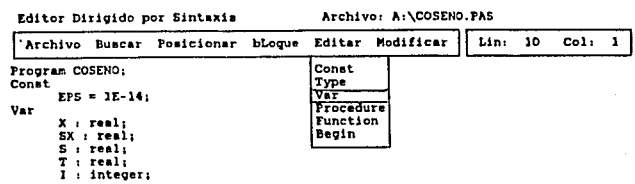


Figura VI.5.

Como el editor sólo puede declarar una variable por línea, hay que introducir cada por medio del siguiente flujo. Al haber seleccionado la palabra Var, el editor espera en primera instancia un identificador para su nombre que, una vez tecleado, escribe el " : " y proporciona una nueva ventana con los diferentes Tipos permitidos, que en nuestro caso corresponde a la opción de Tipo Simple. Figura VI.6.

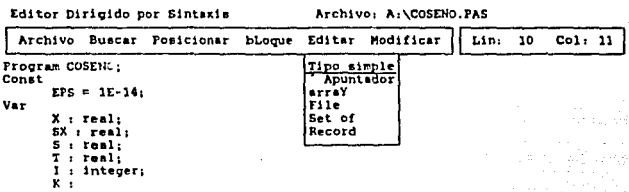


Figura VI.6.

Aquí, es posible declarar en un tipo simple conjuntos y subrangos, pero en nuestro sencillo ejemplo los tipos se reducen a los llamados escalares para lo cual hay que seleccionar en la nueva ventana la opción Tipos, que nos presenta la ventana de los tipos escalares. Figura VI.7.

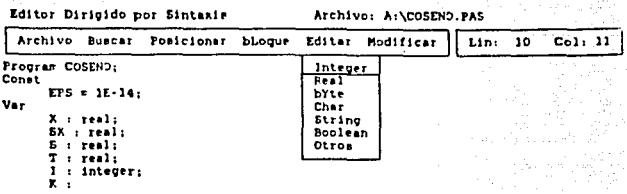


Figura VI.7.

Nótese que aunque es necesaria la declaración por línea, la palabra Var, sólo se aparece al inicio del bloque de variables además de que se genera la identación en forma automática.

Supongamos ahora que en lugar de teclear la variable "s", se teclea nuevamente la ya declarada "sx"; el editor inmediatamente desplegará la ventana de error. Figura VI.8.

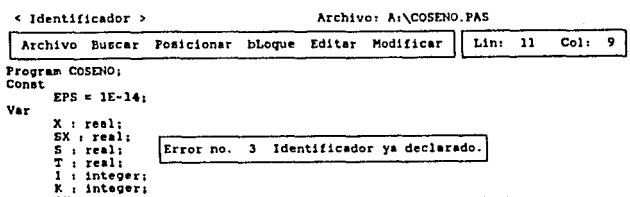


Figura VI.8.

Para quitar el mensaje de error, bastará con oprimir cualquier tecla y el cursor aparecerá nuevamente al final del identificador erroneo, eliminando la "x" con la tecla de retroceso y oprimiendo el retorno, la variable es aceptada como correcta y la ventana de tipos aparecerá en forma normal.

Una vez completadas las declaraciones, para iniciar el programa principal, bastará con posicionarnos en el Begin de la ventana de declaraciones, para dar entrada a la ventana de instrucciones que toma el control en este punto. Figura VI.9.

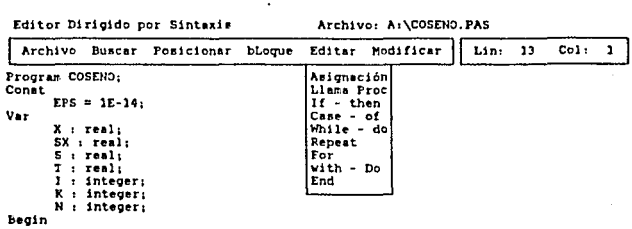


Figura VI.9.

Para el caso de la primera instrucción Read, es considerada por el editor como una llamada a un procedimiento ya que su información ya esta almacenada internamente en la ta-

bla de identificadores.

Para la instrucción For, supondremos que utilizamos la variable no declarada "nd", el editor despliega la ventana indicando el error. Figura VI.10.

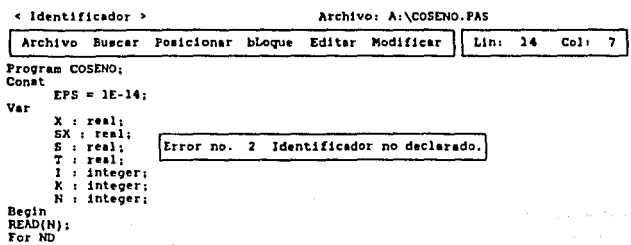


Figura VI.10.

Cuando se corrige el error, automáticamente se genera el ":=" y el editor espera por una expresión que, una vez tecleada y validada da entrada a una pequeña ventana que considera las opciones "to" y "downto", una vez seleccionada la primera opción, el editor pedirá otra expresión que hace terminar la instrucción poniendo automáticamente la palabra reservada "do" y, cumpliendo así estrictamente con la gramática del lenguaje, en la siguiente línea, se inserta su Begin correspondiente, posicionando el cursor debajo de éste con su identación adecuada. Figura VI.11.

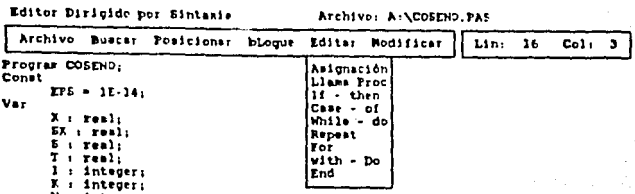


Figura VI.11.

Veamos ahora la expresión t := -t*sx / (k*(k-1)). Si sólo tecleamos hasta la diagonal de división, el editor lo detecta y envía el mensaje de error siguiente. Figura VI.12.

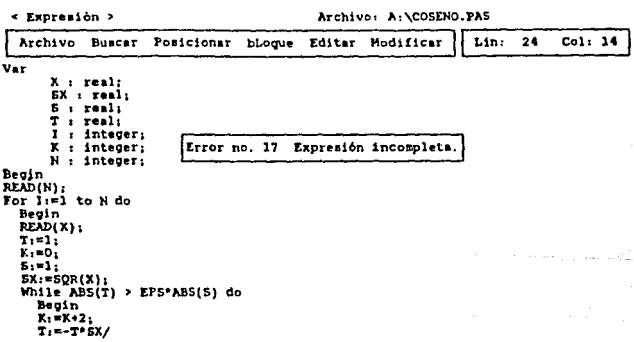


Figura VI.12.

Si completamos la expresión sin el último paréntesis, el editor respondera con un mensaje. Figura VI.13.

```
< Expresión >                          Archivo: A:\COSENO.PAS
 Archivo  Buscar  Posicionar  bLoque  Editar  Modificar | Lin:  24  Col: 22
Var
      X : real;
      SX : real;
      S : real;
      T : real;
      I : integer;
      K : integer;          Error no.  7  Falta  ' ) '.
      N : integer;
Begin
READ(N);
For I:=1 to N do
  Begin
  READ(X);
  T:=1;
  K:=0;
  S:=1;
  SX:=SQR(X);
  While ABS(T) > EPS*ABS(S) do
    Begin
    K:=K+2;
    T:=-T*SX/(K*(K-1)
```

Figura VI.13.

Finalmente al teclear el ")", la expresión será correcta y es posible continuar con la edición. Por último se debe cerrar la instrucción del While y del For, para lo cual se selecciona un End para cada una. El editor pone automáticamente el ";" después del End, o bien un punto al final del programa. Figura VI.14.

```
Editor Dirigido por Sintaxis           Archivo: A:\COSENO.PAS
 Archivo  Buscar  Posicionar  bLoque  Editar  Modificar | Lin:  28  Col:  1
      I : integer;
      K : integer;
      N : integer;
Begin
READ(N);
For I:=1 to N do
  Begin
  READ(X);
  T:=1;
  K:=0;
  S:=1;
  SX:=SQR(X);
  While ABS(T) > EPS*ABS(S) do
    Begin
    K:=K+2;
    T:=-T*SX/(K*(K-1));
    End;
  WRITELN(X,S,K DIV 2);
  End;
End.
```

Figura VI.14.

Como se ve, la edición resulta ser sumamente sencilla y proporciona al programador las herramientas suficientes para la creación de un programa, evitando fundamentalmente errores de tecleo en el caso de pasar directamente el programa del papel a la computadora, o bien errores de lógica sintáctica si es que el programador esta improvisando directamente en la computadora.

VI.2. Resultados de la evaluación.

Los puntos favorables alcanzados por el editor son los siguientes:

- El editor garantiza la generación de programas libres de errores léxicos y sintácticos.

- El tiempo de elaboración de un programa se reduce en comparación con el tiempo utilizado por un editor convencional, el porcentaje de reducción dependerá directamente de las habilidades del programador para acoplarse a la interfase de ventanas llegando en ocasiones hasta un 30%, comparado con un editor de texto común.

- El programador tiene a su disposición una gran cantidad de instrucciones, las cuales estan determinadas por la gramática del lenguaje Pascal y de las cuales gran parte no son conocidas.

- La interface de ventanas facilita un manejo flexible, rápido y eficiente, además de auxiliar al programador en la estructuración del programa.

- Finalmente el editor (EDS) representa una excelente herramienta para el aprendizaje del lenguaje Pascal.

De manera similar se pudieron detectar las siguientes limitantes:

- En el caso de los programadores expertos, trataban de teclear tan rápido que no atendian a las instrucciones del editor produciendo selecciones de funciones o instrucciones no deseadas.

- El no poder manipular o incorporar programas creados por otro editor.

- Finalmente, el tener que esperar a la terminación de el programa, para poder corregir un error que no sea léxico ni sintáctico (errores de lógica, semántica,etc), simplemente errores de tecleo o de una opción mal seleccionada.

Después del análisis de resultados, podemos señalar que el editor (EDS) cumple con los objetivos planteados, abriendo la posibilidad de crear nuevas versiones para eliminar las limitantes detectadas. Una de las mejoras para la cual se han creado las bases, es el manejo semántico de las expresiones que, resulta relativamente fácil agregar por la modularidad del programa.

VII. CONCLUSIONES.

Una de las principales conclusiones a las que se llegó, fue sin duda que el editor dirigido por sintaxis (EDS), indiscutiblemente es una herramienta que beneficia al programador, y no solo al programador experto, sino también al programador nuevo, ya que minimiza el tecleo de caracteres y simplifica la tarea de indentación y estructuración.

También, como característica de cualquier editor dirigido por sintaxis, el EDS genera todas las palabras claves así como los simbolos de la gramática, de tal forma que todo programa creado por éste, se almacena sin errores de sintáxis. Esto proporciona una ayuda más, para el posterior paso de compilación haciendose ésta mas eficiente, pues es posible detectar la naturaleza del error con mayor facilidad, incluyendo que desaparece la posibilidad de que existan errores léxicos y sintácticos que, como ya se ha dicho, representan hasta el 90% del total de errores de una primera compilación.

Es necesario hacer énfasis en que, sin lugar a dudas, el EDS no dificulta en lo más mínimo ninguna etapa de la compilación o edición sino que por el contrario ayuda en gran medida a optimizar éstas funciones.

Finalmente, podemos señalar que la interfase con el usuario obtenida, es flexible, sencilla y eficiente, lo cual representa para el programador, una gran ventaja al no tener que auxiliarse necesariamente de un manual para su utilización.

G L O S A R I O

GLOSARIO.

Acceso : Indica una posibilidad o un método para que se comuniquen partes de una computadora o para transferir datos.

Alfanumérico : Contracción de ALFAbético y NUMERICO. Se denomina asi a un conjunto de caracteres constituido por letras, números y/o caracteres especiales. No se consideran alfanuméricos los caracteres de control ni los semigráficos.

ANALEX : Analizador léxico, también conocido como Scanner. Es el procedimiento que analiza los aspectos léxicos de un lenguaje.

ANASIN : Analizador sintáctico o Parser. Es el procedimiento que analiza la sintaxis de un determinado lenguaje.

Análisis Léxico : Estudio de la escritura correcta u ortografía de las palabras de un lenguaje.

Análisis Semántico : Estudio del significado de las palabras de un lenguaje. Cuando la computadora intenta comprender las ordenes o instrucciones de un programa realiza un análisis semántico.

Análisis Sintáctico : Estudio de las relaciones en un leguaje para comprobar si corresponden a las reglas gramaticales de éste.

Arbol : Estructura de datos utilizada para representar niveles. Formalmente es un conjunto de uno o mas nodos en el que hay un nodo especial, llamado raiz, y los demás nodos son particiones en subconjuntos disjuntos.

Archivo : Conjunto ordenado de registros organizados de cierta manera para facilitar su acceso. En general, pueden contener millones de datos por lo que se almacenan en memoria secundaria.

Area de Memoria : Parte de la memoria central caracterizada por tener direcciones contiguas.

Arreglo : Indica una tabla vector o matriz que agrupa un conjunto homogeneo de variables.

ASCII : Abreviatura de American Standard Code for Information Interchange. Representa un código de siete bits para la transmisión o almacenamiento de caracteres. Se suele extender a ocho bits con fines prácticos para completar un byte ya sea con un bit de paridad o para extender el conjunto de caracteres permitidos hasta 256.

Asignación : Atribución de un valor a una variable. Se logra mediante las instrucciones de asignación provistas por los lenguajes de programación.

Autómata : Diagrama gráfico de representación de las estructuras gramaticales de un determinado lenguaje. Sirve para verificar que una oración pertenece a la sintaxis de un lenguaje definido.

Bloque : Es una secuencia contigua de caracteres dentro de un texto. Marcando un bloque, el usuario de un editor habilita operaciones a ser ejecutadas con la pieza de texto completa como una unidad.

BNF : Backus Normal Form, (forma normal de Backus). Simbolismo adecuado para representar las estructuras sintácticas de un lenguaje.

Borrar : Operación mediante la cual se elimina un dato, o bien se libera para nuevas grabaciones, una zona de memoria. Desde el punto de vista físico no es necesario borrar o suprimir un dato, sino que es suficiente impedir la posibilidad de leerlo o escribirlo encima de otro.

Buffer : Area de memoria de almacenamiento temporal, que facilita el manejo inmediato de datos.

Búsqueda : Acción de buscar un dato o cualquier cosa. Se habla de búsqueda de un registro en un archivo, de un sector o una pista en un disco o bien de un dato en cualquier estructura de datos (tablas, listas, árboles, etc.).

Cadena : Sucesión o cadena de caracteres pertenecientes a un determinado alfabeto.

Caracter : Cualquiera de los símbolos representados por un código (por ejemplo ASCII). Ejemplos de caracteres son las letras del alfabeto, las cifras numéricas, los signos de puntuación o los caracteres gráficos.

Caracteres gráficos : Son caracteres especiales cuya forma permite componer dibujos. Estos caracteres, denominados también semigraficos son controlados por la computadora como cualquier otro caracter alfanumérico. Su conjunto forma una cadena al igual que las letras.

Codificación : Escritura de un programa en un lenguaje de programación. Proceso de traducción de un código.

Código : Conjunto de símbolos y reglas que sirven para representar datos o instrucciones, el conjunto de todos los símbolos posibles de un determinado código toma el nombre de alfabeto, y mediante este alfabeto es posible contruir palabras y frases de dicho código. Los códigos tienen el objeto de convertir la información a una forma mas adecuada para su transmisión y/o proceso. Ejemplo de un código es el código Morse, que permitía la transmisión de caracteres en las lineas telegráficas.

Columna : Indica un conjunto de datos dispuestos en sentido vertical. Se habla de columnas de una matriz así como del número de caracteres susceptibles de impresión en impresora o visualizables en una linea de pantalla.

Comando : Término con el que se suele indicar una instrucción dada a la computadora. No obstante, es conveniente reservar esta denominación para las ordenes dadas directamente por el teclado (tales como: run, list, control-C, etc.) o desde el sistema operativo y utilizar el término de instrucción para las que se dan a través de un programa (tales como print o goto). Un comando puede modificar texto o bien manipular bloques, archivos o ventanas.

Compilador : Programa adecuado para traducir el texto de un programa escrito en un lenguaje de alto nivel a su versión ejecutable por la computadora. Algunas veces los compiladores traducen directamente a lenguaje de máquina, y otras, a lenguaje ensamblador que tendrá que ser ensamblado por el compilador correspondiente.

Conjunto : Grupo o asociación de caracteres o palabras bajo una misma característica.

Constante : Valor que representa un dato, de cualquier tipo, definido inicialmente en un programa y ya no modificado.

Cursor : Es un pequeño bloque o linea en la pantalla (algunas veces intermitente), que marca el lugar en donde se esta realizando la edición. En un editor de pantalla el cursor esta usualmente dentro de una ventana.

Dato : Término genérico para indicar números, letras u otros caracteres existentes en una computadora, o en su memoria y sobre los cuales actúan los programas.

Debug : Literalmente significa espulgar e indica todas las técnicas de búsqueda y corrección de errores en un programa. Suele encontrarse mas amenudo como depurar.

Despachador de comandos : Es un procedimiento dentro de un editor (o de cualquier programa interactivo) que interpreta caracteres tecleados como comandos, y llama a uno o mas procesadores de comandos para ejecutar dichos comandos.

Dirección : Número o nombre que permite identificar las posiciones individuales de una memoria. Parte de una instrucción que especifica en que posición de memoria esta contenido el operando.

Directorio : Tabla utilizada para direccionar de forma directa, los archivos contenidos en una memoria secundaria. Va grabada en el mismo soporte de memoria (dico duro ó disketes). El directorio contiene los nombres de los archivos gra-

bados y la información necesaria para tener acceso a cada uno de ellos. Con frecuencia, el directorio se completa con toda la informatición relativa a cada archivo individual.

Display : Término inglés que, como sustantivo, indica un medio de presentación visual como la pantalla. Como verbo, se refiere al proceso de visualización o presentación de datos.

Drive : Parte mecánica de una memoria de disco flexible o de cinta magnética, que pone en rotación el disco o la cinta. Con frecuencia, con fines prácticos, indica todo el dispositivo físico en el que se inserta un diskette, es decir, la unidad perferica completa.

Editor : Programa que permite al usuario, crear, actualizar o modificar información, usualmente almacenada en archivos. Algunos editores son usados para editar gráficos. Otros editan información textual codificada en un lenguaje entendible por un computador.

Editor dirigido por sintaxis (EDS) : Editor que tiene la peculiaridad de no permitir errores de sintaxis durante la creación o modificación de un programa. Se basa en la gramática de un determinado lenguaje.

Editor de texto : Editor que crea, actualiza y modifica archivos que contienen información textual codificada. Usual-

mente el editor no sabe si el texto es un programa, una carta u otro tipo de estructura.

Ensamblador : Programa compilador necesario para traducir el texto de un programa escrito en lenguaje ensamblador a lenguaje de máquina. El lenguaje ensamblador es un lenguaje de bajo nivel, muy similar al lenguaje de máquina. A cada una de sus instrucciones corresponde una sola instrucción de la máquina por lo que es característico de cada computador.

Error : Término utilizado para denotar una situación no permitida por un lenguaje o un sistema operativo. Cuando se produce un error en una computadora no se obtienen los resultados esperados.

Estructura de datos : Conjuto de datos dispuestos según determinadas reglas que permiten un control en el manejo de la información.

Etiqueta : Indica un conjunto de caracteres utilizados como dirección o referencia para el reconocimiento de una posición o valor.

Expresión : Conjunto de operadores y operandos. Las expresiones pueden ser algebráicas o booleanas.

FIFO : First Input First Output (Primeras entradas, primeras salidas), método para servir a una cola de espera según la cual el primero que llega es el primero en ser atendido.

Función : Correlación o vínculo entre una o mas variables independientes y una variable dependiente. En informática el concepto de función es análogo al concepto matemático. En los lenguajes de programación algunas funciones estan definidas mientras que otras pueden ser creadas por el programador con instrucciones.

Grafo : Conjunto de elementos organizados entre los cuales existe una relación.

Gramática : Reglas especficas que determinan el orden y forma de los elementos de un lenguaje.

Hash : Técnica de direccionamiento de un elemento en una estructura en la que la dirección se calcula con la aplicación de algún criterio matemático (función de hash) a partir de un valor de la llave. La palabra hash, en inglés no significa nada por lo que no tiene una traducción al español.

Inicialización : Procedimiento que establece los valores iniciales necesarios para un funcionamiento correcto. Generalmente los lenguajes de programación inicializan las variables con cero para numéricas y nulo para las alfabéticas.

Instrucción : Orden singular impartida por una computadora mediante un programa escrito en un determinado lenguaje.

Lenguaje : conjunto de caracteres y reglas bien definidas sobre cuya base es posible escribir un programa de computadora.

LIFO : Last Input First Output (Ultimas entradas, primeras salidas) método para servir una cola de espera en donde los últimos que entran son los primeros en salir.

Linea : Como significado habitual indica una linea de texto o renglón. A menudo indica una linea de programa.

Lista : Estructura de datos que consta de un número variable de nodos.

Marca : Caracter señalador utilizada como aviso de que se verifica determinada situación. En una edición indica la posición del texto, generalmente usada para agrupar un bloque.

Matriz : Tabla bidimensional de valores. El término en inglés que determina a una matriz es ARRAY.

Memoria central : Parte de la computadora que contiene los programas en ejecución y los datos sobre los que operan. Se le suele denominar RAM (Random Acces Memory). Esta constituí-

da por un conjunto de celdas iguales asociadas a una cierta dirección.

Memoria de lectura : Conocida como ROM (Read Only Memory), es una memoria que sólo puede ser leída y que contiene datos permantes.

Memoria secundaria : Unidad periférica adecuada para almacenar grandes cantidades de datos tal como los discos duros, diskettes, cartuchos, cintas, etc.

Menú : Lista de procedimientos entre los que el usuario puede escoger el deseado.

Operador : Entidad que define el tipo de acción a desarrollar con los operandos. Los principales operadores se distribuyen en aritméticos(+,-,*,/), lógicos (AND, OR) y de relación (>,<,=,<>).

Operando : Argumento sobre el que actúa un operador.

Overflow : Indica la superación de los valores máximos permitidos ya sea de la memoria o de alguna estructura de datos fija.

Overlay : Técnica de aprobechamiento de la memoria central, que permite cargar, en tiempos diferentes, partes distintas de un mismo programa. Tiene el objeto de permitir la ejecu-

ción de programas demasiado grandes con objeto con respecto a la memoria disponible.

Página : Por diversos motivos, la memoria puede distribuirse en módulos iguales que toman el nombre de páginas. En la edición, es un conjunto de líneas que conforman la pantalla.

Palabra reservada : Palabra reconocida como código de instrulcción en determinado lenguaje de programación.

Pantalla : (Screen o Display) Parte de la unidad de video terminal o monitor en donde aparecen las imagenes.

Parámetro : Variable independiente que toma valores fijos. Dato solicitado por un programa o subrutina, para condicionar su desarrollo o para elegir entre diferentes modos de funcionamiento.

Párser :Ver Anasin.

Pascal : Lenguaje de programación de alto nivel cuyo nombre recuerda al matemático francés B. Pascal. Escrito originalmente en 1971 se ha consolidado por sus caracteristicas de precisión formal y por la posiblilidad de estructurar los programas.

Pila : Conocida en inglés como stack, es un conjunto de registros o posiciones de memoria que se organiza según la téc-

nica denominada LIFO, los datos introducidos en una pila se extraen luego en orden inverso al de escritura.

Procedimiento : Secuencia de operaciones que han de realizarse para desarrollar un cierto cometido o para emplear un determinado proceso de datos.

Procesador de comandos : Procedimiento que asocia el trabajo actual con un determinado comando en ejecución.

Procesador de texto : Programa que se utiliza para modidicar, escribir y transformar textos. Cuando el texto es lenguaje natural se conoce como procesador de palabras.

Promt : Indicador de presencia o caracter de aviso. Símbolo utiilizado por la computadora o algún lenguaje para avisar sobre algunas situaciones particulares. El cursor puede ser considerado como promt.

Programa fuente : Programa escrito en un lenguaje de alto o bajo nivel, antes de la compilación.

Programa objeto : Programa obtenido después de la compilación o ensamblado de un programa fuente, esta constituido por instrucciones de máquina y puede ser ejecutado por la computadora, aunque en ocasiones es necesario unirlo con utilerias o subrutinas (ligarlo) para que pueda ser ejecutable.

RAM : Random Access Memory, memoria a cuyos elementos individuales se puede tener acceso en el mismo tiempo de forma indistinta. Se ha impuesto la costumbre de llamar así a una memoria en donde es posible desarrollar operaciones tanto de escritura como de lectura.

Recursividad : Se aplica a un proceso de cálculo repetitivo que "se llama a si mismo" (con diversos parámetros) en el curso del cálculo. En los lenguajes estructurados es posible escribir programas recursivos.

ROM : Read Only Memory, dispositivo adecuado para conservar información y realizado con tecnología de integración a gran escala. Esta caracterizada por el hecho de que el usuario solo puede leer cuando esta contenido en la memoria y nunca modificarlo. La grabación se realiza en el momento de su construcción en la fábrica.

Salvar : Indica una acción de transferencia a una memoria permanente de datos o programas contenidos en memoria principal. Casi siempre las memorias en donde se salva son unidades de disco.

Scanner : Ver Analex.

Scroll : Operación de desplazamiento de la imagen de la pantalla hacia arriba (up) o hacia abajo (down).

Semántica : Estudio del significado de las palabras en los lenguajes naturales. En los lenguajes de programación se habla de semántica a la comprensibilidad de un programa por parte de la computadora, es decir, la posibilidad de traducirlo a lenguaje de máquina para su ejecución.

Símbolo : Indica cualquier cosa a la que se atribuye un significado convencional. En informática indica uno o varios caracteres a los que se atribuye un significado, como pueden ser las variables y palabras reservadas.

Sintaxis : Conjunto de reglas sobre cuya base se construye un lenguaje, siendo la relación que deben guardar las palabras y los símbolos para establecer una instrucción.

Sistema Operativo : Conjunto de programas base de una computadora que contolan y administran en forma óptima todas las partes y recursos.

Software : Conjunto de parogramas de una computadora.

Stack : (ver pila).

Tabla : Sinónimo de vector o matriz (ARRAY). Conjunto de elementos constituidos por, almenos dos campos, clave e información.

Template : Esqueleto o plantilla sintáctica que contiene las palabras clave y los signos de puntuación dados por una instrucción.

Texto : Secuencia de caracteres o lineas que estan siendo editadas. El texto puede ser un programa, un documento o cualquier secuencia de uno o mas caracteres.

Token : Elemento mínimo de reconocimiento de un Anasin. El token puede ser un identificador, un número, un operador, etc. Los tokens son identificados por el Analex.

Top-down : Término en inglés que significa "desde arriba hacia abajo", que se utiliza para indicar la elaboración de programas a partir de una visión general, pasando luego a definir situaciones cada vez mas concretas. Se utiliza un técnica top-down cuando se escriben previamente bloques de programa mas externos y se pasa luego a los internos.

Usuario : Persona que utiliza la computadora y a quién van dirigidos los nuevos desarrollos.

Variable : Nombre simbólico utilizado para representar una magnitud cuyo valor efectivo puede variar durante la ejecución de un programa.

Variables Globales : Variables de cualquier tipo, que pertenecen simultanteámente a todos los bloques de un programa,

o a un programa y sus subrutinas.

Variables Locales : Variables utilizadas solamente en el ámbito de un programa, se su bloque, o de una de sus subrutinas.

Ventana : Región de la pantalla de la computadora, utilizada para guardar información en forma separada de la restante que puede ser presentada en la pantalla al mismo tiempo.

B I B L I O G R A F I A

BIBLIOGRAFIA.

[1] Garlan-Miller, GNOME: An Introductory Programming Enviroment Based on a Family of Structure Editors, ACM-Sigplan V.19, No.5, mayo 1984.

[2] Hansen, W. J., Creation of hieratic text with a computer display, Computer Science, Stanford University. junio 1971.

[3] Teitelbaum-Reps, The Cornell Program Synthesizer. ACM-Sigplan, V. 24, No. 9, septiembre 1981. pp.563-573.

[4] Habermann, A. N., D. Notkin. the Gandalf Software Development Environment, technical report, Computer Science, Carnegie-Mellon University, enero 1982.

[5] Campbell-Kirslis, The SAGA Project: A System for Software Development, ACM-Sigplan V.19, No. 5, mayo 1984. pp. 71-80.

[6] Harrison-Rosenfeld, Structured editing with RPDE Graphics, mayo 1986. pp. 93-101.

[7] Giacalone Alessandro, Rinard M. C. Doeppner Thomas, IDEOSY An Ideographic and Interactive Program Description System ACM-Sigplan V.19, No. 5, mayo 1984.

[8] Zelkowitz Marvin, A Small Contribution to Editing with a Syntax Directed Editor. ACM-Sigplan, V. 19, No. 5, Mayo 1984. pp. 1-6.

[9] Delisle Norman, Menicosy David, Schwartz Mayer D. Viewing a Programming Environment as a single tool. ACM-Sigplan, V. 19, No. 5, mayo 1984. pp. 49-56.

[10] Donzeau,Gouge,Kahn,Lang,Mélese, Document Structure and Modularity in Mentor, ACM-Sigplan V.19, No. 5, May 1984. pp. 141-148.

[11] Fischer,Johnson,Mauney,Pal,Stock. The Poe Language-Based Editor Project, ACM Sigplan V.19, #5, May 1984. pp 21-29.

[12] Archer James, Conway Richard. COPE a Cooperative Programming Environment. Junio 1981.

[13] Zavodnick R. J. Middleton M. D. YALE, The design of yet another Language-based Editor,ACM- SIGPLAN V.21, No. 16, 1986. pp. 70-78.

[14] Wirth Niklaus, Algorithms + Data Structures = Programs, U.S.A., Prentice-Hall, 1976.

[15] Medina Mora,Feiler. An Incremental Programming Environment. IEEE TRansactions on Software Engineering, V. 7, No. 5, Septiembre 1981.pp. 472-482.

[16] Fritzson, Peter, Preliminary Experience from the DICE system a Distributed Incremental Compiling Environment, ACM-Sigplan V.19, #5, May 1984.

[17] Wilander J. An interactive Programming System for Pascal. BIT V.20, No. 2, 1980. pp. 163-174.

[18] Teitelman W. Interlisp reference manual. Xerox 1978.

[19] Webster Bruce, A simple windowing system. Byte, V. 11, No.3, marzo 1986. pp. 129-133.

[20] Tremblay-Sorenson, Compiler Writing, McGraw Hill, U.S.A 1985.pp.796.

[21] Comer Douglas, Shen Vincent. Hash-Bucket Search. Software Practice and Experience. V.2, 1982. pp. 669-681.

[22] Torres Octavio, El usuario de editores de texto: Un análisis de sus errores, Cero uno cero, Mayo 1984.

[23] Miller L., Thomas J., Behavioral issues in the use of interactive systems. International journal of man machine. V. 9 1977.

[24] Van Dam A., Rice D., On line text editin: a surver. Comp uting surveys. V. 3 Sep. 1971.

[25] Scapin D., Computer commands in restricted natural languajes. Human factors, V. 23 1981.

[26] Leadgard H., Whiteside J., The natural languaje of interactive systems. Comm. ACM, V. 23 1980.

[27] Roberts T., Moran T., The evaluation of text editors, Comm. ACM, V. 26, abril 1983.

[28] Rosemberg J., Evaluating the suvestiggeness of command names. Proceding of the conference of human factors in computer systems. Marzo 1982.

[29] Standish-Taylor, Arcturus: A Prototype Advanced Ada Programming Environment, ACM-Sigplan V.19, #5, May 1984.

[30] Horgan-Moore, Techniques for Improving Language-Based Editors, ACM-Sigplan V.19, #5, May 1984.

[31] Steven P. Reiss, Graphical Program Development with PECAN Program Development System, ACM-Sigplan, V.19, No. 5, May 1984.

[32] Chesi-Dameri-Franceschi-Gatti-Simonelli, ISDE: An Interactive Software Development Environment, ACM-Sigplan V.19, #5, May 1984.

[33] Wood, Steven R., Z-The 95% Program Editor, ACM-Sigplan V.19, #5, May 1984.

[34] Jensen K. y Wirth N. Pascal, User Manual and Report, Berlín, Springer-Verlag, 1974.

[35] Tenenbaum, A. y Augenstein, M., Data Structures Using Pascal, U.S.A., Prentice-Hall, 1982.

[36] Welsh, J. y Elder, J., Introduction to Pascal, U.S.A., Prentice-Hall, 1982.

[37] Knuth, Donald E., Fundamentals Algorithms, The Art of Computer Programming, Vol.1. Mass U.S.A, Addison-Wesley 1973.

[38] Knuth, Donald E., The Art of Computer Programming, V.2, Mass, U.S.A., Addison-Wesley, 1973.

[39] Knuth, Donald E., Sorting and Searching, The Art of Computer Programming.Vol.3 U.S.A. Addison-Wesley 1973.

[40] Donovan, John, Systems Programming, New York, U.S.A., McGraw-Hill, 1972.

[41] Euán A.- Cordero Luis, Estructuras de datos, Universidad Nacional Autónoma de México. Facultad de Ingeniería. México 1982.

[42] Teitelbaum-Reps-Horwitz. The Why and Wherefore of the Cornell Program Synthesizer. ACM-Sigplan, V. 16, No. 4, mayo 1981.pp. 8-16.

[43] Fraser, Syntax-Directed Editing of General Data Structures. ACM-Sigplan, V. 16, No. 4, mayo 1981. pp.17-21.

[44] Edwars Charles. Advanced Techniques in Turbo Pascal. SYBEX Books. pp. 311.

[45] Legarreta Garciadiego Luis. Compiladores. Fundación Arturo Rosenblueth. pp. 663.

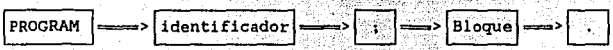
[46] Anklam Patricia, Cutler David, Heinen Roger. Engineering a Compiler VAX-11 Code Generation. Digital. pp. 269.

[47] Barrett William A. Compiler construction. Theory and practice. Science Research Associates Inc. pp. 661.

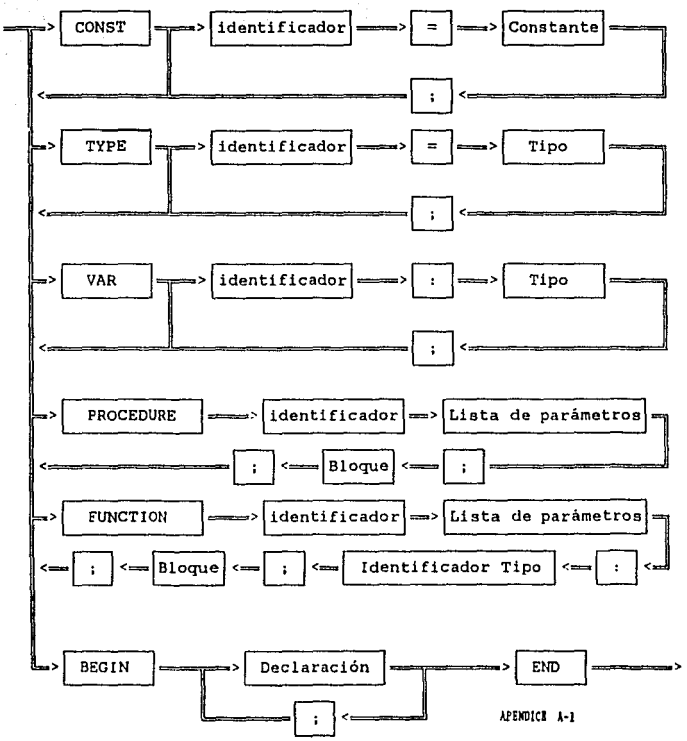
A P E N D I C E A

SINTAXIS DEL LENGUAJE PASCAL.

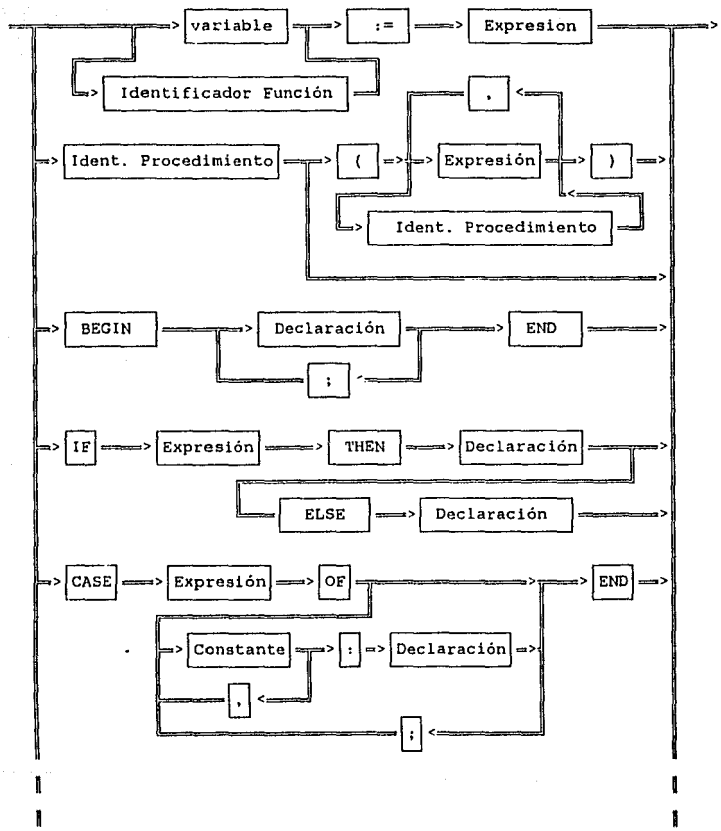
Program :



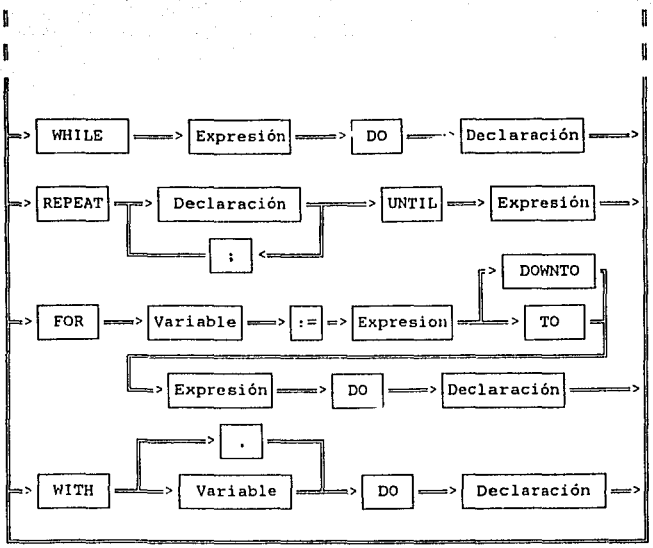
Bloque :



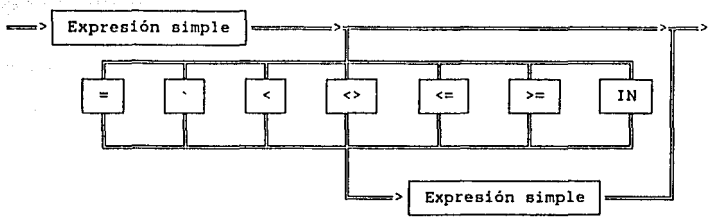
Declaración :



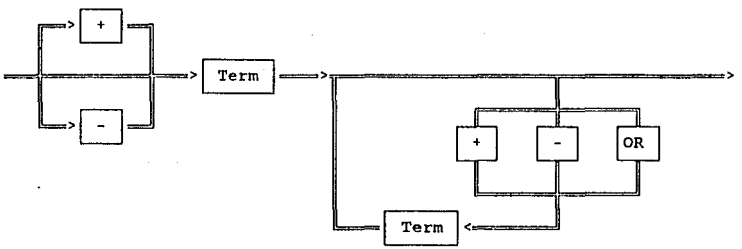
Declaración : (continuación)



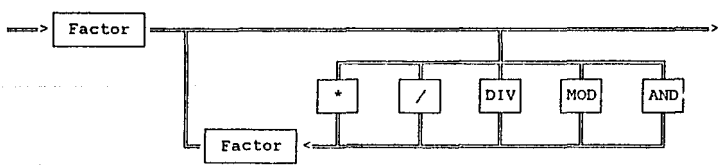
Expresión :



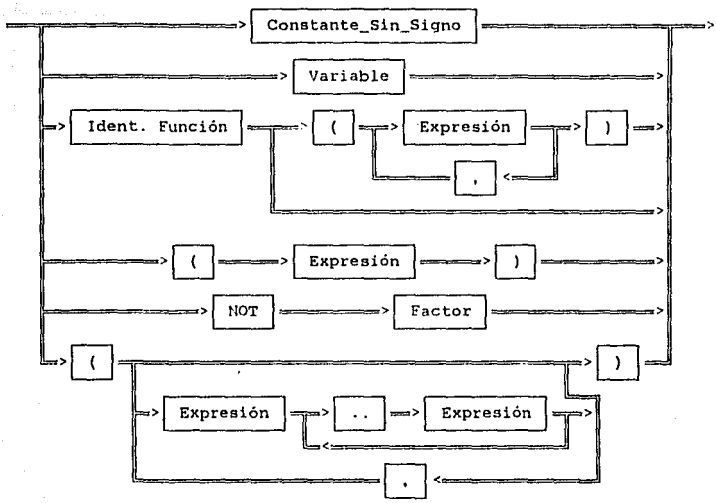
Expresión simple :



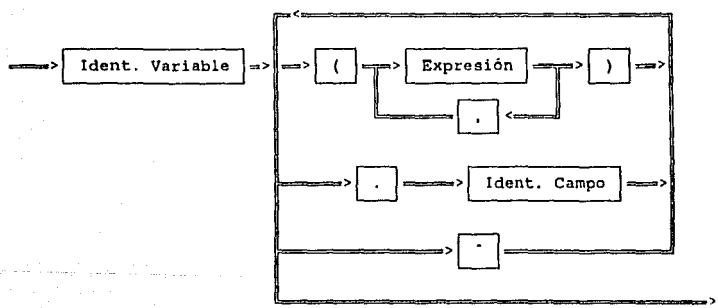
Term :



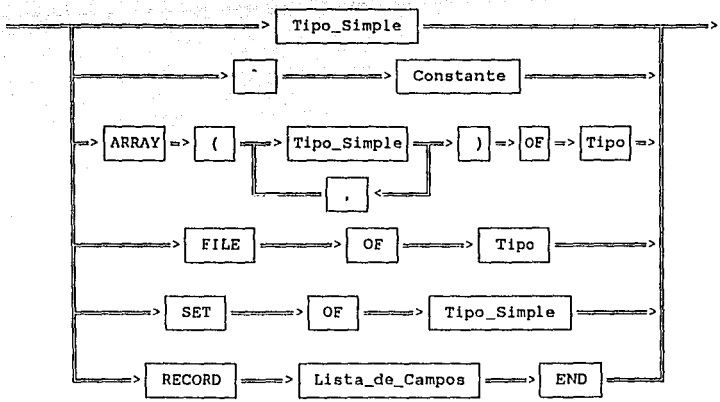
Factor:



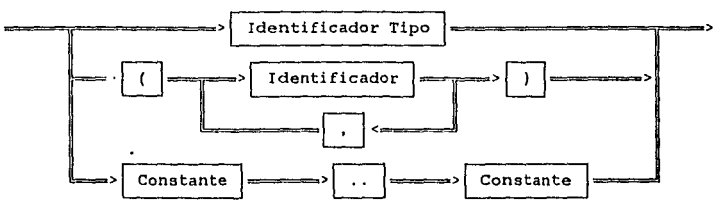
Variable :



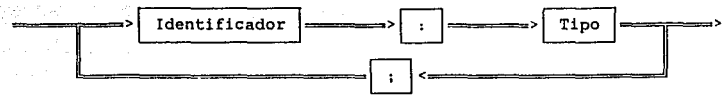
Tipo :



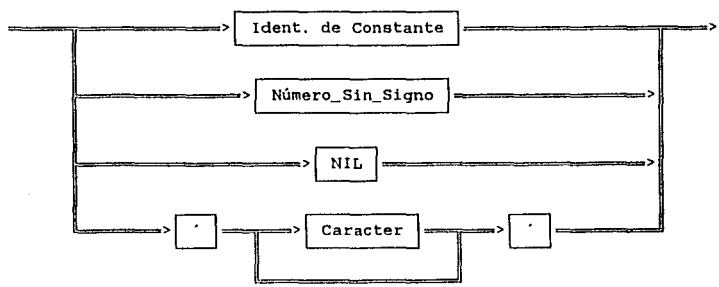
Tipo_Simple :



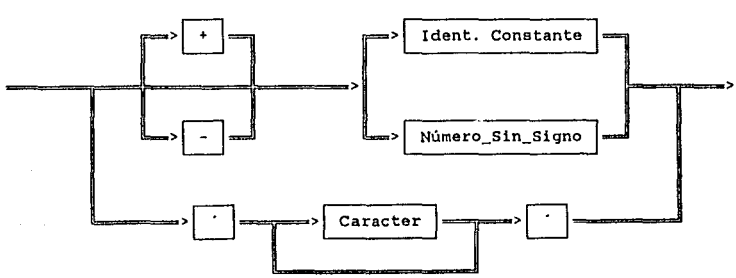
Lista_de_Campos :



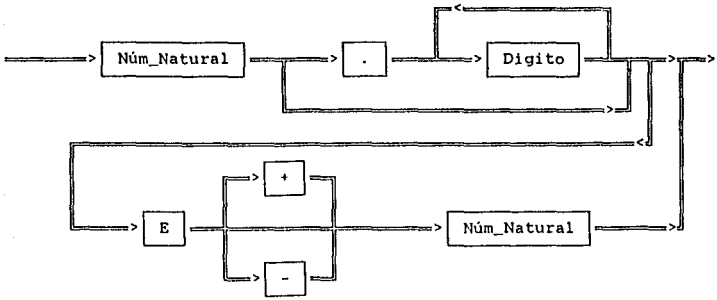
Constante_Sin_Signo :



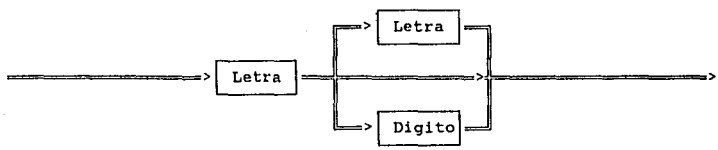
Constante :



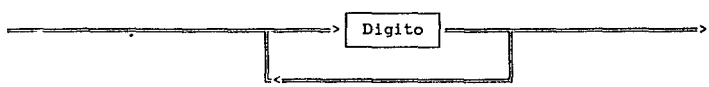
Número_Sin_Signo :



Identificador :



Núm_Natural :



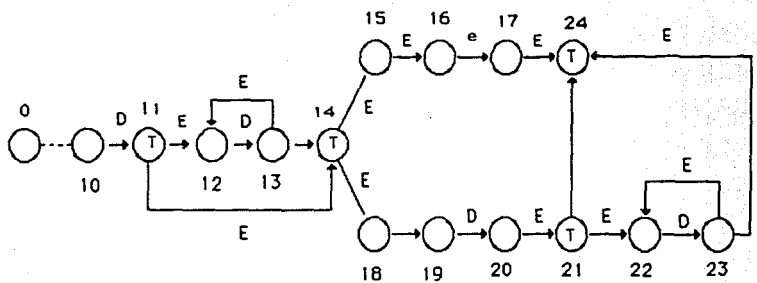
A P E N D I C E B

1.- IDENT

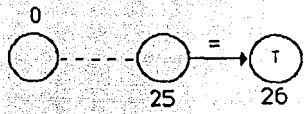
L(L/D)*

2.- NUM

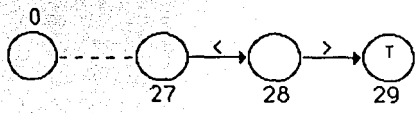
DD*(E|D·D*)



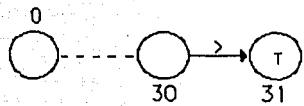
3.- '='



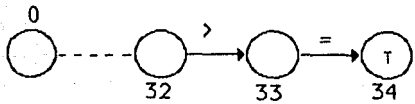
4.- '<>'



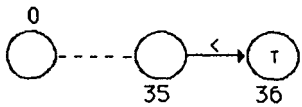
5.- '>'



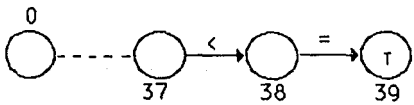
6.- '>='



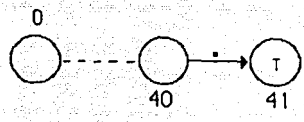
7.- '<'



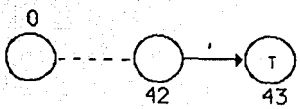
8.- '<='



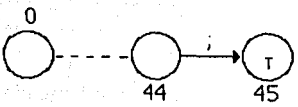
9.- '.'



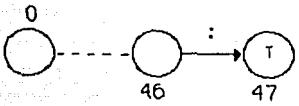
10.- ','



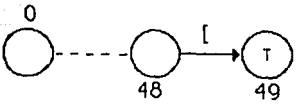
11.- ';'



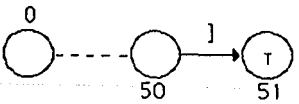
12.- ':'



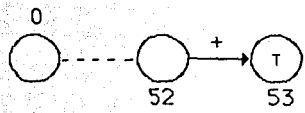
13.- '['



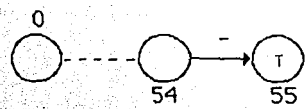
14.- ']'



15.- '+'

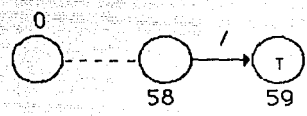


16.- '-'

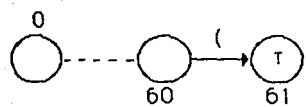


17.- '*'

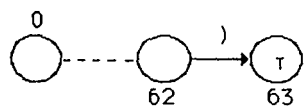
18.- '/'



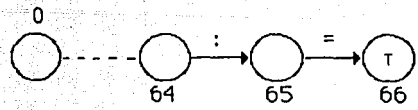
19.- '('



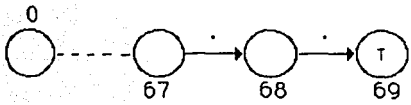
20.- ')'



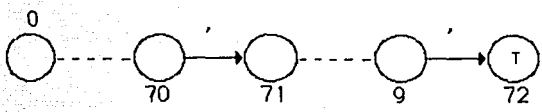
21.- ':= '



22.- '..'



23.- ' ' '



A P E N D I C E C

LISTA DE TOKENS UTILIZADOS EN EL EDS.

No. de token	Token	No. de token	Token
1	Identificador	41	STRING
2	Entero	42	BOOLEAN
3	Real	43	BEGIN
4	<	44	END
5	>	45	IF
6	.	46	THEN
7	,	47	ELSE
8	:	48	DO
9	;	49	WHILE
10	[	50	FOR
11	]	51	TO
12	+	52	DOWNTO
13	-	53	REPEAT
14	*	54	UNTIL
15	/	55	FILE
16	(	56	OF
17	)	57	SET
18	<>	58	ARRAY
19	<=	59	FUNCTION
20	>=	60	PROCEDURE
21	..	61	CASE
22	:=	62	RECORD
23	'Constante'	63	WITH
24	^		
25	=		
26	NOT		
27	DIV		
28	MOD		
29	AND		
30	OR		
31	NIL		
32	IN		
33	PROGRAM		
34	VAR		
35	TYPE		
36	CONST		
37	INTEGER		
38	REAL		
39	BYTE		
40	CHAR		

TABLA I.

TABLA DE ESTADOS ORIGINAL

Edo	Letra	Digito	=	<	>	.	,	:	;	[	]	+	-	*	/	(	)	^	"	Bl.
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	31	18	1	0
1	19	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0
2	0	21	0	0	0	-27	0	0	0	0	0	0	0	0	0	0	0	0	0	30	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	23	0	22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	19	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0
20	19	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0
21	0	21	0	0	0	-27	0	0	0	0	0	0	0	0	0	0	0	0	0	-30	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-30	0
28	0	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-30	0
29	0	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-30	0
30	0	2	0	0	0	0	0	0	0	0	0	2	2	0	0	0	0	0	0	0	0
31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	0	31	31	31

0 -Representa el estado nulo.
+N -Representa un estado terminal.
-N -Representa un estado no terminal.

TABLA II.

TABLA DE ESTADOS OPTIMIZADA

Edo	Letra	Digito	=	<	>	.	,	;	:	[	]	+	-	*	/	(	)	^	~	E	Bl.
0	1	2	3	4	5	6	3	3	7	3	3	3	3	3	3	3	3	15	3	1	0
1	8	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0
2	0	10	0	0	0	-11	0	0	0	0	0	0	0	0	0	0	0	0	0	-14	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	3	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	8	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0
9	8	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0
10	0	10	0	0	0	-11	0	0	0	0	0	0	0	0	0	0	0	0	0	-14	0
11	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-14	0
12	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-14	0
13	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-14	0
14	0	2	0	0	0	0	0	0	0	0	0	2	2	0	0	0	0	0	0	0	0
15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	0	15	15	15

0 -Representa el estado nulo.
+N -Representa un estado terminal.
-N -Representa un estado no terminal.

TABLA III.

FUNCIONES Y PROCEDIMIENTOS DEL EDS.

Nombre	Clase	Tipo	Parametros
READ	P	0	0
READLN	P	0	0
WRITE	P	0	0
WRITELN	P	0	0
ABS	F	1	1
ARCTAN	F	2	1
COS	F	2	1
EXP	F	2	1
FRAC	F	2	1
INT	F	2	1
LN	F	2	1
SIN	F	2	1
SQR	F	2	1
SQRT	F	2	1
APPEND	P	0	2
ASSIGN	P	0	2
CLOSE	P	0	1
ERASE	P	0	1
RESET	P	0	1
REWRITE	P	0	1
SEEK	P	0	2
EOF	F	6	1
EOLN	F	6	1
SEEKEOLN	F	6	1
SEEKEEOF	F	6	1
DISPOSE	P	0	1
NEW	P	0	1
UPCASE	F	4	1
ODD	F	6	1
PRED	F	0	1
SUCC	F	0	1
CLRSCR	P	0	0
GOTOXY	P	0	2
DELETE	P	0	3
INSERT	P	0	3
STR	P	0	2
VAL	P	0	3
COPY	F	5	3
LENGTH	F	1	1
POS	F	1	2
CHR	F	4	1
ORD	F	1	1
ROUND	F	1	1
TRUNC	F	1	1

TABLA IV.

LISTA DE LOS ERRORES DETECTADOS POR EL EDS.

Error no. 1 Identificador ilegal.
Error no. 2 Identificador no declarado.
Error no. 3 Identificador ya declarado.
Error no. 4 Constante ilegal.
Error no. 5 Falta ' , '.
Error no. 6 Falta ' ('.
Error no. 7 Falta ') '.
Error no. 8 Falta ' [' o ' ('.
Error no. 9 Falta '] '.
Error no. 10 Falta " ' ".
Error no. 11 Falta ' . '.
Error no. 12 Falta ' .. '.
Error no. 13 Palabra reservada.
Error no. 14 Diferencia de tipos.
Error no. 15 Sobreflujo en tabla de identificadores.
Error no. 16 Caracter inválido.
Error no. 17 Expresión incompleta.
Error no. 18 Identificador no es constante.
Error no. 19 Tipo no definido.
Error no. 20 Tipo invalido.
Error no. 21 El identificador debe ser variable o función.
Error no. 22 Procedimiento no definido.
Error no. 23 El identificador debe ser una variable.
Error no. 24 Campo no declarado.
Error no. 25 Existe un campo adicional en la expresión.
Error no. 26 Identificador no válido en la expresión.
Error no. 27 La función no lleva parámetros.
Error no. 28 Faltan parámetros.
Error no. 29 El archivo no existe.
Error no. 30 Archivo no abierto para entrada.
Error no. 31 Archivo no abierto para salida.
Error no. 32 El archivo no esta abierto.
Error no. 33 Error de lectura o directorio no encontrado.
Error no. 34 Error de escritura en el disco.
Error no. 35 Formato númerico no válido.
Error no. 36 No permitido sobre dispositivo lógico.
Error no. 37 No permitido en modo directo.
Error no. 38 No se puede asignar a archivo estándard.
Error no. 39 Diferente longitud de registro.
Error no. 40 No hay búsqueda después de EOF.
Error no. 41 Inesperado fin de archivo.
Error no. 42 No disponible para cerrar archivo origen.
Error no. 43 Disco lleno.
Error no. 44 Sobreflujo en el tamaño del archivo.
Error no. 45 No disponible para cerrar archivo destino.
Error no. 46 Ruta de directorio inválido.
Error no. 47 No hay memoria disponible para continuar.
Error no. 48 El bloque no ha sido definido.
Error no. 49 El bloque no se puede desplegar.

LISTA DE LOS ERRORES DETECTADOS POR EL EDS. (Continuación...)

Error no. 50	Demasiados archivos abiertos.
Error no. 51	La cadena no se encuentra.
Error no. 52	Error en la creación del archivo.
Error no. 53	Archivo demasiado grande, lectura parcial.
Error no. 54	No disponible para crear archivo ".BAK".
Error no. 55	Error en generaciòn de ligas.
Error no. 56	Archivo IST no corresponde con archivo PAS.
Error no. 57	En esta linea no hay opción a Marcar Bloque.
Error no. 58	No se permite copiar/mover bloque de Instr. a Declaración
Error no. 59	No esta permitido copiar/mover en esa linea.
Error no. 60	No se permite copiar/mover bloque de Declaración a Instr.
Error no. 61	No se permite borrar en bloque de Declaración.
Error no. 62	No se puede editar, ya existe un programa.
Error no. 63	No hay modificaciòn, no existe algún programa.
Error no. 64	No esta permitido insertar en esta línea.
Error no. 65	Sobreescritura no válida, provoca rompimiento de Sintaxis.

TABLA V.

DECLARACION DE LA INFORMACION DE VENTANAS.

MENU PRINCIPAL

```
  st01 : string[08] = ' Archivo';
  st02 : string[08] = '  Buscar';
  st03 : string[12] = '  Posicionar';
  st04 : string[08] = '  bLoque';
  st05 : string[08] = '  Editar';
  st06 : string[11] = '  Modificar';
  st07 : string[01] = '';
```

SUBMENUS ASOCIADOS.

Menú Pull-Down # 1 (Manejo de Archivos)

```
  st11 : string[10] = 'Salvar    ';
  st12 : string[09] = 'salvar En';
  st13 : string[10] = 'Directorio';
  st14 : string[10] = 'cAmbia dis';
  st15 : string[08] = 'Imprimir';
  st16 : string[06] = 'Copiar';
  st17 : string[09] = 'Renombrar';
  st18 : string[06] = 'Borrar';
  st19 : string[05] = 'saLir';
  st1x : string[01] = '';
```

Menú Pull-Down #2 (Comandos de Búsqueda)

```
  st21 : string[10] = 'Busca     ';
  st22 : string[09] = 'Siguiente';
  st2x : string[01] = '';
```

Menú Pull-Down #3 (Comandos de Posicionamiento)

```
  st31 : string[17] = 'Inicio de archivo';
  st32 : string[14] = 'Fin de archivo';
  st33 : string[16] = 'inicio de Bloque';
  st34 : string[13] = 'fin de bloQue';
  st35 : string[05] = 'Linea';
  st3x : string[01] = '';
```

Menú Pull-Down #4 (Comandos de Bloque)

```
  st41 : string[07] = 'Marca  ';
  st42 : string[05] = 'Copia';
  st43 : string[05] = 'mueVe';
  st44 : string[07] = 'Suprime';
  st4x : string[01] = '';
```

Menú Pull-Down #5 (Ventana de Edición)

```
  st51 : string[08] = 'Program ';
  st52 : string[05] = 'Salir';
  st5x : string[01] = '';
```

DECLARACION DE LA INFORMACION DE VENTANAS. (Continuación...)

Menú Pull-Down #6 (Ventana de Modificación)

```
  st61 : string[13] = 'Sobreescribir';
  st62 : string[08] = 'Insertar';
  st6x : string[01] = '';
```

VENTANAS DE DIRECCION DE SINTAXIS.

Grupo de Palabras Reservadas # 1

```
  sg11 : string[09] = 'Const    ';
  sg12 : string[04] = 'Type';
  sg13 : string[03] = 'Var';
  sg14 : string[09] = 'Procedure';
  sg15 : string[08] = 'Function';
  sg16 : string[05] = 'Begin';
  sg1x : string[01] = '';
```

Grupo de Palabras Reservadas # 2

```
  sg21 : string[10] = 'Asignación';
  sg22 : string[10] = 'Llama Proc';
  sg23 : string[09] = 'If - then';
  sg24 : string[09] = 'Case - of';
  sg25 : string[10] = 'While - do';
  sg26 : string[06] = 'Repeat';
  sg27 : string[03] = 'For';
  sg28 : string[09] = 'with - Do';
  sg29 : string[03] = 'End';
  sg2x : string[01] = '';
```

Grupo de Palabras Reservadas # 3

```
  sg31 : string[05] = '  ,  ';
  sg32 : string[03] = '  ;';
  sg3x : string[01] = '';
```

Grupo de Palabras Reservadas # 4

```
  sg41 : string[05] = '  ,  ';
  sg42 : string[03] = '  :';
  sg4x : string[01] = '';
```

Grupo de Palabras Reservadas # 5

```
  sg51 : string[16] = ' Con parámetros ';
  sg52 : string[15] = ' Sin parámetros';
  sg5x : string[01] = '';
```

DECLARACION DE LA INFORMACION DE VENTANAS. (Continuación...)

```
Grupo de Palabras Reservadas # 6

  sg61 : string[10] = 'Por valor ';
  sg62 : string[03] = 'Var';
  sg63 : string[08] = 'Function';
  sg64 : string[09] = 'proCedure';
  sg6x : string[01] = '';

Grupo de Palabras Reservadas # 7

  sg71 : string[05] = '  ;  ';
  sg72 : string[03] = '  )';
  sg7x : string[01] = '';

Grupo de Palabras Reservadas # 8

  sg81 : string[05] = '  ,  ';
  sg82 : string[03] = '  )';
  sg8x : string[01] = '';

Grupo de Palabras Reservadas # 9

  sg91 : string[08] = 'eTiqueta';
  sg92 : string[04] = 'eLse';
  sg93 : string[03] = 'End';
  sg9x : string[01] = '';

Grupo de Palabras Reservadas # 10

  sgA1 : string[08] = 'Con else';
  sgA2 : string[08] = 'Sin else';
  sgAx : string[01] = '';

Grupo de Palabras Reservadas # 11

  sgB1 : string[11] = 'Tipo simple';
  sgB2 : string[11] = '^ Apuntador';
  sgB3 : string[05] = 'arraY';
  sgB4 : string[04] = 'File';
  sgB5 : string[06] = 'Set of';
  sgB6 : string[06] = 'Record';
  sgBx : string[01] = '';

Grupo de Palabras Reservadas # 12

  sgC1 : string[05] = '  ,  ';
  sgC2 : string[03] = '  ]';
  sgCx : string[01] = '';

Grupo de Palabras Reservadas # 13

  sgD1 : string[05] = '  ;  ';
  sgD2 : string[04] = ' End';
  sgDx : string[01] = '';
```

DECLARACION DE LA INFORMA(.ON DE VENTANAS. (Continuación...)

```
Grupo de Palabras Reservadas # 14

  sgE1 : string[08] = '   To   ';
  sgE2 : string[07] = ' Downto';
  sgEx : string[01] = '';

Grupo de Palabras Reservadas # 15

  sgF1 : string[09] = '   Tipo  ';
  sgF2 : string[09] = ' Conjunto';
  sgF3 : string[09] = ' Subrango';
  sgFx : string[01] = '';

Grupo de Palabras Reservadas # 16

  sgG1 : string[09] = ' Integer ';
  sgG2 : string[05] = ' Real';
  sgG3 : string[05] = ' bYte';
  sgG4 : string[05] = ' Char';
  sgG5 : string[07] = ' String';
  sgG6 : string[08] = ' Boolean';
  sgG7 : string[06] = ' Otros';
  sgGx : string[01] = '';
```

TABLA VI.