2 ej.
15

Universidad Nacional Autónoma de México

FACULTAD DE CIENCIAS

Desarrollo de un Compilador para un Lenguaje Orientado a Resolver Problemas Estadísticos

T E S I S

Que para obtener el Título de:

A C T U A R I O

P R E S E N T A :

Carlos Alberto Esquivel Avila

MEXICO, D. F. 1986.

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

DERECHOS RESERVADOS © PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis está protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

2 ej.
15

Universidad Nacional Autónoma de México

FACULTAD DE CIENCIAS

Desarrollo de un Compilador para un Lenguaje Orientado a Resolver Problemas Estadísticos

T E S I S

Que para obtener el Título de:

A C T U A R I O

P R E S E N T A :

Carlos Alberto Esquivel Avila

MEXICO, D. F. 1986.

I N D I C E

Pág.

I. INTRODUCCION 1

II. CAMBIOS A LA IMPLANTACION ORIGINAL DEL COMPILADOR 5

III. ESTRUCTURA DEL COMPILADOR

A) ANALIZADOR LEXICOGRAFICO Y SINTACTICO 12

B) ANALIZADOR SEMANTICO Y GENERACION DE CODIGO 13

C) PROGRAMAS DE UTILERIA 13

IV. PROGRAMAS DE UTILERIA

A) HT7. PROCESADOR DE LA GRAMATICA 15

B) HPC2. GENERADOR DEL DICCIONARIO DE PALABRAS CLAVE 18

V. ANALIZADOR LEXICOGRAFICO Y SINTACTICO

A) INTRODUCCION 20

B) ANALIZADOR LEXICOGRAFICO 24

C) ANALIZADOR SINTACTICO 26

1) PRESENTACION Y JUSTIFICACION 26

2) CARACTERISTICAS PRINCIPALES 29

D) CAMBIOS EFECTUADOS AL ANALIZADOR LEXICOGRAFICO Y SINTACTICO 33

E) PRINCIPALES ESTRUCTURAS DE DATOS Y RUTINAS DEL RECONOCEDOR 38

VI. ANALIZADOR SEMANTICO Y GENERACION DE CODIGO

A) INTRODUCCION 42

B) ESTRUCTURA Y FUNCIONAMIENTO DEL ANALIZADOR SEMANTICO 42

C) LA TABLA DE SIMBOLOS 47

1) ORGANIZACION DE LA TABLA DE SIMBOLOS 47

2) DESCRIPTORES 52

Pág.

D) ADMINISTRACION DE MEMORIA 59

1) STACK DE MEMORIA 59

2) REPRESENTACION DE VARIABLES 62

E) COMPILACION DE EXPRESIONES 68

1) INTRODUCCION 68

2) OPERACIONES BINARIAS Y UNARIAS 69

3) STACKS DE OPERANDOS Y OPERADORES 72

4) EVALUACION DE OPERACIONES EN EL PROGRAMA OBJETO 76

5) STACK DE DESCRIPTORES DE OPERANDOS EN EL PROGRAMA OBJETO 78

6) ARREGLOS EXPLICITOS 80

7) EXPRESION SELECTIVA DE ARREGLO 84

F) PROPOSICION DE ASIGNACION 106

G) PROPOSICION CONDICIONAL 110

H) PROPOSICION DE TRANSFERENCIA INCONDICIONAL 126

I) RESUMEN DE MODIFICACIONES AL ANALIZADOR SEMANTICO 135

J) PRINCIPALES ESTRUCTURAS DE DATOS Y RUTINAS DEL ANALIZADOR SEMANTICO 141

K) PRINCIPALES ESTRUCTURAS DE DATOS Y RUTINAS DEL PROGRAMA OBJETO 145

VII. MODIFICACIONES Y MEJORAS PROPUESTAS AL COMPILADOR 150

VIII. CONCLUSION 157

AGRADECIMIENTOS 160

APENDICES

A) GRAMATICA DEL LENGUAJE 161

I) CONVENCIONES PARA ESCRIBIR LA GRAMATICA

II) LISTADO DE LA GRAMATICA

B) LISTA DE ERRORES EN COMPILACION 168

C) LISTA DE ERRORES EN EJECUCION 173

D) LISTA DE LLAMADOS SEMANTICOS 175

E) EJEMPLOS 177

F) MANUAL DE REFERENCIA Y OPERACION 193

BIBLIOGRAFIA 198

I N D I C E

		Pág.
I.	INTRODUCCION	1
II.	CAMBIOS A LA IMPLANTACION ORIGINAL DEL COMPILADOR	5
III.	ESTRUCTURA DEL COMPILADOR	
	A) ANALIZADOR LEXICOGRAFICO Y SINTACTICO	12
	B) ANALIZADOR SEMANTICO Y GENERACION DE CODIGO	13
	C) PROGRAMAS DE UTILERIA	13
IV.	PROGRAMAS DE UTILERIA	
	A) HT7. PROCESADOR DE LA GRAMATICA	15
	B) HPC2. GENERADOR DEL DICCIONARIO DE PALABRAS CLAVE	18
V.	ANALIZADOR LEXICOGRAFICO Y SINTACTICO	
	A) INTRODUCCION	20
	B) ANALIZADOR LEXICOGRAFICO	24
	C) ANALIZADOR SINTACTICO	26
	1) PRESENTACION Y JUSTIFICACION	26
	2) CARACTERISTICAS PRINCIPALES	29
	D) CAMBIOS EFECTUADOS AL ANALIZADOR LEXICOGRAFICO Y SINTACTICO	33
	E) PRINCIPALES ESTRUCTURAS DE DATOS Y RUTINAS DEL RECONOCEDOR	38
VI.	ANALIZADOR SEMANTICO Y GENERACION DE CODIGO	
	A) INTRODUCCION	42
	B) ESTRUCTURA Y FUNCIONAMIENTO DEL ANALIZADOR SEMANTICO	42
	C) LA TABLA DE SIMBOLOS	47
	1) ORGANIZACION DE LA TABLA DE SIMBOLOS	47
	2) DESCRIPTORES	52

Pág.

D) ADMINISTRACION DE MEMORIA 59

1) STACK DE MEMORIA 59
2) REPRESENTACION DE VARIABLES 62

E) COMPILACION DE EXPRESIONES 68

1) INTRODUCCION 68
2) OPERACIONES BINARIAS Y UNARIAS 69
3) STACKS DE OPERANDOS Y OPERADORES 72
4) EVALUACION DE OPERACIONES EN EL PROGRAMA OBJETO 76
5) STACK DE DESCRIPTORES DE OPERANDOS EN EL PROGRAMA OBJETO 78
6) ARREGLOS EXPLICITOS 80
7) EXPRESION SELECTIVA DE ARREGLO 84

F) PROPOSICION DE ASIGNACION 106
G) PROPOSICION CONDICIONAL 110
H) PROPOSICION DE TRANSFERENCIA INCONDICIONAL 126
I) RESUMEN DE MODIFICACIONES AL ANALIZADOR SEMANTICO 135
J) PRINCIPALES ESTRUCTURAS DE DATOS Y RUTINAS DEL ANALIZADOR SEMANTICO 141
K) PRINCIPALES ESTRUCTURAS DE DATOS Y RUTINAS DEL PROGRAMA OBJETO 145

VII. MODIFICACIONES Y MEJORAS PROPUESTAS AL COMPILADOR 150

VIII. CONCLUSION 157

AGRADECIMIENTOS 160

APENDICES

A) GRAMATICA DEL LENGUAJE 161

I) CONVENCIONES PARA ESCRIBIR LA GRAMATICA

II) LISTADO DE LA GRAMATICA

B) LISTA DE ERRORES EN COMPILACION 168
C) LISTA DE ERRORES EN EJECUCION 173
D) LISTA DE LLAMADOS SEMANTICOS 175
E) EJEMPLOS 177
F) MANUAL DE REFERENCIA Y OPERACION 193

BIBLIOGRAFIA 198

DESARROLLO DE UN COMPILADOR PARA UN LENGUAJE ORIENTADO A RESOLVER PROBLEMAS ESTADISTICOS

I. INTRODUCCION

El presente trabajo nació de la necesidad del Grupo de Estadística, de la Facultad de Ciencias de la UNAM, de contar con recursos computacionales que realmente satisfagan sus requerimientos académicos y de investigación.

Se puede decir que la participación de la computación como soporte de trabajo en el campo de la estadística matemática ha sido escasa en comparación con la que ha tenido en otros campos de estudio.

Los paquetes estadísticos no siempre presentan beneficios para el estadístico. Desde el punto de vista del investigador, el paquete es un producto rígido, pues no se puede modificar, ni agregar al contenido, procedimientos, salidas y opciones del mismo. Además frecuentemente resulta muy engorroso tratar de ligar la entrada o la salida del paquete a un lenguaje de programación. Así, el usuario debe adaptarse a los recursos y li-

mitaciones de los lenguajes y paquetes disponibles, habiendo ocasiones en que debe repetir trabajo de captura de datos.

Desde el punto de vista académico, los paquetes también presentan desventajas. Dado que el paquete emite únicamente los resultados de los procedimientos pedidos, el estudiante tiene nula participación en la manipulación y análisis de los datos. Se deja de lado el punto de interés de la materia, consistente en que el alumno comprenda y domine los algoritmos de análisis estadístico.

De los lenguajes de programación que gozan de popularidad, pocos son los que ofrecen herramientas realmente útiles al estadístico.

Según las necesidades de los estadísticos, un lenguaje que los auxilie, debe permitir el manejo de grandes volúmenes de datos y por ende, tener procesos eficientes y sencillos de lectura y escritura de datos. También debe permitir emitir reportes sofisticados, como son gráficas, histogramas y tablas y poseer las herramientas para que el usuario cree sus propios reportes. Además debe ofrecer procesos sencillos para manipular matemáticamente los datos, como son operaciones, entre matrices, comparaciones entre matrices, obtención de cualesquiera submatrices, traspuestas, inversas y determinantes de matrices bidimensionales.

Los lenguajes de programación de propósitos generales, que son los más conocidos, carecen de muchas de las características mencionadas, por lo que cada vez que es necesario elaborar nuevos programas o modificarlos, se requiere invertir mucho tiempo.

Desde el punto de vista académico, el alumno no puede prescindir de la computadora, debido a la cantidad de datos que debe manipular. Sin embargo, el recurir a un lenguaje de programación, ocasiona que el alumno invierta mucho tiempo en programar y depurar sus algoritmos, sobretodo si el lenguaje no posee

DESARROLLO DE UN COMPILADOR PARA UN LENGUAJE ORIENTADO A RESOLVER PROBLEMAS ESTADISTICOS

I. INTRODUCCION

El presente trabajo nació de la necesidad del Grupo de Estadística, de la Facultad de Ciencias de la UNAM, de contar con recursos computacionales que realmente satisfagan sus requerimientos académicos y de investigación.

Se puede decir que la participación de la computación como soporte de trabajo en el campo de la estadística matemática ha sido escasa en comparación con la que ha tenido en otros campos de estudio.

Los paquetes estadísticos no siempre presentan beneficios para el estadístico. Desde el punto de vista del investigador, el paquete es un producto rígido, pues no se puede modificar, ni agregar al contenido, procedimientos, salidas y opciones del mismo. Además frecuentemente resulta muy engorroso tratar de ligar la entrada o la salida del paquete a un lenguaje de programación. Así, el usuario debe adaptarse a los recursos y li-

mitaciones de los lenguajes y paquetes disponibles, habiendo ocasiones en que debe repetir trabajo de captura de datos.

Desde el punto de vista académico, los paquetes también presentan desventajas. Dado que el paquete emite únicamente los resultados de los procedimientos pedidos, el estudiante tiene nula participación en la manipulación y análisis de los datos. Se deja de lado el punto de interés de la materia, consistente en que el alumno comprenda y domine los algoritmos de análisis estadístico.

De los lenguajes de programación que gozan de popularidad, pocos son los que ofrecen herramientas realmente útiles al estadístico.

Según las necesidades de los estadísticos, un lenguaje que los auxilie, debe permitir el manejo de grandes volúmenes de datos y por ende, tener procesos eficientes y sencillos de lectura y escritura de datos. También debe permitir emitir reportes sofisticados, como son gráficas, histogramas y tablas y poseer las herramientas para que el usuario cree sus propios reportes. Además debe ofrecer procesos sencillos para manipular matemáticamente los datos, como son operaciones, entre matrices, comparaciones entre matrices, obtención de cualesquiera submatrices, traspuestas, inversas y determinantes de matrices bidimensionales.

Los lenguajes de programación de propósitos generales, que son los más conocidos, carecen de muchas de las características mencionadas, por lo que cada vez que es necesario elaborar nuevos programas o modificarlos, se requiere invertir mucho tiempo.

Desde el punto de vista académico, el alumno no puede prescindir de la computadora, debido a la cantidad de datos que debe manipular. Sin embargo, el recurrir a un lenguaje de programación, ocasiona que el alumno invierta mucho tiempo en programar y depurar sus algoritmos, sobretodo si el lenguaje no posee

herramienta como las mencionadas. La consecuencia final es que el estudiante se aleja del punto de interés de la materia, que es el estudio de la estadística, para concentrarse en actividades propias de un programador profesional.

Por todas las razones expuestas, se decidió desarrollar un lenguaje de programación orientado a la estadística, que cumpliera las características mencionadas y que auxiliara tanto al investigador como al estudiante y profesor de estadística.

El diseño del lenguaje quedó a cargo de miembros del Grupo de Computación del Departamento de Matemáticas, de la misma facultad. Ese trabajo quedó plasmado en la tesis de licenciatura de Javier García García [G1] y se recomienda leerla para poder entender la filosofía del lenguaje cuya implantación se describe aquí.

El lenguaje diseñado también tiene la característica de ser de propósitos generales, posee la estructura de bloques en que viven ALGOL y PASCAL y también tiene estructuras de control semejantes a las de aquellos.

En este trabajo se presenta la implantación del compilador del lenguaje en una microcomputadora.

Entre las principales características del compilador vale mencionar que el análisis sintáctico está dirigido por tablas, permitiendo modificar fácilmente la gramática, sin tener que corregir el analizador sintáctico. El análisis semántico y la generación de código son dirigidos por medio de llamados a rutinas semánticas, que vienen contenidos en el mismo árbol sintáctico.

El compilador debe considerarse más bien como un traductor, pues el código generado corresponde a otro lenguaje de alto nivel. De esta forma, para poder ejecutar el programa, se debe compilar antes el código. Sobra decir que el lenguaje objeto debe estar

instalado en la computadora de trabajo.

Como la implantación del lenguaje se está haciendo en una microcomputadora, para facilitar la instalación y la transportación a otra máquina, se decidió generar lenguaje de alto nivel, evitando así tener que construir un intérprete para cada implantación y sólo se requerirán algunos cambios en la generación de código, esencialmente en lo relacionado al manejo de archivos.

Además, el código final es más eficiente, pues tiene la garantía de eficiencia y optimización del código generado por el compilador del programa pseudo-objeto.

Debido a que durante el diseño del compilador, se presentó un problema de saturación de la memoria en la computadora anfitriona (que sólo cuenta con 64K), se decidió dividir al compilador en dos fases. La primera parte se encarga de leer el programa fuente y aplicar los análisis lexicográfico y sintáctico, produciendo un conjunto de tablas que se depositan en disco. La segunda etapa se ocupa de efectuar el análisis semántico y la generación de código a partir de aquellas tablas.

Hasta ahora, esta estructura del compilador ha dado buen resultado.

II. CAMBIOS A LA IMPLANTACION ORIGINAL DEL COMPILADOR

En un principio, el compilador se estaba implantando en una computadora NOVA 3/12, que ofrece una capacidad en memoria principal y secundaria equiparable a la de una microcomputadora.

Dicha máquina se encuentra instalada en el laboratorio de Estadística de la Facultad de Ciencias y dispone de los lenguajes FORTRAN y BASIC.

Tomando en cuenta los pocos recursos de esta máquina, se decidió emplear a FORTRAN, tanto para el lenguaje anfitrión, como para el lenguaje objeto. Además, se ocupó a BASIC para desarrollar dos programas de utilería, que procesan a la gramática y a un diccionario de palabras clave y entregan un conjunto de tablas que son tomadas por los reconocedores sintáctico y semántico. Se utilizó a BASIC en estos casos debido a que FORTRAN adolece de serias limitaciones para almacenar y operar cadenas de caracteres.

Después de considerar los problemas de trabajar en esas condi-

ciones, se hicieron algunos cambios y actualmente la máquina empleada es la microcomputadora FRANKLIN ACE-1200, la cual es compatible con algunos modelos de APPLE. También se cambió a PASCAL, tanto en el lenguaje anfitrión, como en el lenguaje objeto y los dos programas de utilería también se implantaron en PASCAL.

La serie de motivos que se presentaron para llevar a cabo estas rotundas determinaciones se enumeran a continuación:

1) Existía el proyecto de eliminar la máquina NOVA del laboratorio de Estadística y sustituirla por otra máquina.
   Ya en la actualidad, se dispone de una máquina adicional, una ALTOS, bastante más poderosa, pero a la que no se ha podido transmitir información desde la NOVA.*
   Tener que editar nuevamente todos los programas y archivos de datos involucrados en el proyecto, así como corregirlos para adecuarlos a las versiones existentes de FORTRAN y BASIC, presenta una perspectiva verdaderamente desalentadora, que atrasaría el avance del proyecto por un gran tiempo o bien, quedaría el compilador como un producto terminado, pero en una máquina obsoleta.
   Por otra parte la computadora ALTOS contiene un compilador de PASCAL y para pasar los programas fuentes del compilador estadístico a ella se requiere escencialmente de dos pasos. El primer paso consiste en transmitir los fuentes a una computadora PC. Este proceso no es muy complicado y puede hacerse en la misma Facultad. El segundo paso reside en dejar los fuentes en un disco flexible de PC y llevarlo a la computadora ALTOS, la que puede leer y escribir a discos flexibles de PC.
   Con este procedimiento y haciendo algunos cambios a los pro

---

* Esta máquina se adquirió hasta después de haber cambiado la implantación a FRANKLIN y por eso, ya no se tomó en cuenta para este trabajo.

gramas (si son necesarios), se podrá instalar el compilador estadístico en la ALTOS.

2) La máquina NOVA requiere de una estricta estabilidad en las condiciones de temperatura y humedad del medio ambiente. Debido a tal vulnerabilidad, la máquina puede emitir resultados erróneos que pasen desapercibidos para el usuario, errores fatales que suspenden a la computadora e inclusive se pueden llegar a dañar los discos flexibles y el disco duro.

3) La computadora NOVA dispone para memoria secundaria de una unidad de disco duro y de dos de disco flexible, de ocho pulgadas, un cuarto.
La entrada y salida a discos flexibles es mucho más lenta que en una microcomputadora (como la APPLE o la PC) y más fácilmente se saturan de información, por lo que se vuelve sumamente complicado y tedioso su uso para programas muy grandes y para manejar grandes volúmenes de datos.
Por otra parte el disco duro, cuya velocidad de acceso es bastante más rápida que en los discos flexibles y el espacio disponible nunca se ha llegado a saturar, es sumamente delicado y en varias ocasiones se ha dañado, perdiendo por completo la información almacenada.

4) La máquina NOVA carece de impresora. En vez de ella, se posee una terminal de papel que trabaja en paralelo con la terminal de pantalla. Así tanto la salida a consola como la impresión resultan bastante lentas comparándolas con lo que ocurre en una microcomputadora (como la APPLE o la PC) y una impresora ATI-ARGOS.

5) La máquina NOVA permite trabajar sólo a un usuario. Tomando en cuenta que las salidas a pantalla e impresora son muy lentas, provocan que el acceso a la máquina por un grupo numeroso de usuarios sea muy limitado.

6) La Facultad de Ciencias cuenta con alrededor de veinte microcomputadoras FRANKLIN ACE-1200, destinadas a estudiantes y con otras dos, para el uso de los profesores del departamento.
Entre las principales ventajas de emplear estas máquinas para desarrollar el proyecto, cabe mencionar que son compatibles con las máquinas APPLE, que gozan de enorme popularidad. Estas máquinas no requieren de aire acondicionado, el acceso a discos flexibles es mucho más rápido que en la NOVA y respecto a la eficiencia de administración de espacio en disco, ocurre lo mismo.
La velocidad de salida a pantalla es bastante más rápida que en la NOVA y cuentan con varias impresoras ATI-ARGOS.
Es muy importante mencionar que aunque las máquinas pertenecen a toda la Factultad, el acceso a éstas, por parte de la gente ligada al área de estadística será tanto o más fluido, que para la NOVA y la ALTOS.

7) El monto en memoria principal disponible para un programa es mayor en las micros que en la NOVA, por alrededor de 1K. Esto se estimó empleando FORTRAN en la NOVA y la versión PASCAL 1.1 en FRANKLIN. Esta versión sólo puede manipular 64K. Cabe mencionar que los modelos de APPLE: APPLE-IIE y APPLE-IIC, disponen de 128K y que la versión PASCAL 1.2 puede tener acceso a los 128K y que es compatible con la versión 1.1 .
Además, si el compilador se transportara a computadoras PC, ya sea al UCSD PASCAL (que es el mismo compilador utilizado en FRANKLIN) o a otro compilador, como el TURBO PASCAL (en el que se requerirían hacer algunos cambios), la memoria disponible podría ser hasta de 640K.

8) Las versiones de FORTRAN y BASIC de que dispone la NOVA son muy rudimentarias (por ejemplo los identificadores en BASIC constan a lo más de dos caracteres). La búsqueda y depuración de errores en FORTRAN es muy escabrosa y tedio-

sa, debido a que la información que proporciona el compila dor es escasa y deficiente.
El editor de la NOVA (un editor por caracter), también sufre de serias deficiencias por los pobres recursos de presentación, corrección y manipulación de textos.

9) Las ventajas que presenta un lenguaje de bloques como PASCAL sobre FORTRAN y BASIC, para desarrollar programas no-numéricos, como un compilador, son evidentes.
PASCAL tiene mayor facilidad para ver a un mismo dato con varias máscaras (como real, como enteros o como otro tipo de variable); tiene estructuras de control más poderosas; un programa puede realmente modularizarse; contempla la recursividad; permite un más fácil desarrollo, depuración, corrección y lectura de programas; es más flexible y en conclusión más apto para implantar un compilador, que FORTRAN y BASIC.
Cabe indicar que el lenguaje BASIC fué empleado en la NOVA para escribir los programas encargados de procesar a la gramática y al diccionario de palabras clave del lenguaje, debido a que estos programas se basan en el manejo de cadenas y FORTRAN no ofrece flexibilidad alguna en este aspecto.
El UCSD PASCAL ofrece un igual o mejor manejo de cadenas y además es un lenguaje más poderoso.
Además la creciente popularidad que va adquiriendo PASCAL, permite la posibilidad de transportar el compilador a muchas otras máquinas.

10) El hecho de que todos los programas que intervienen en el proyecto estén escritos en PASCAL, le da mayor solidez y homogeneidad al compilador.

11) En la decisión de designar como lenguaje objeto a PASCAL y no a FORTRAN o a un lenguaje ensamblador se confrontaron varios aspectos que se presentan a continuación:

Al generar lenguaje de alto nivel, en primer lugar me libero

de tener que construir un intérprete, el cual tiene que depurarse y corregirse conforme crezca el compilador, o bien evito el trabajo de generar código correspondiente al ensamblador del PASCAL de UCSD y de todas las implicaciones que surgen de tener la responsabilidad de controlar totalmente a la máquina. Así el código generado no tiene que ser tan minucioso, ni tan extenso.

La contraparte aparece principalmente en la falta de dominio que el compilador tiene sobre el código final, quedando supeditado al código generado por el lenguaje objeto.

Sin embargo, es más fácil generar lenguaje de alto nivel y es más fácil modificar la generación de código y se aprovecharía automáticamente el incremento en eficiencia logrado por nuevas versiones de PASCAL UCSD.

Pascal es un lenguaje mucho más robusto que FORTRAN, pudiéndose generar código estructurado, eficiente, más legible y más fácil de depurar.

Pascal puede ver una misma área de memoria como real, como enteros o como caracteres, cualidad que sólo algunas versiones de FORTRAN contemplan, mediante la instrucción EQUIVALENCE, la cual es mucho menos flexible y manipulable que la multiplicidad de tipos en PASCAL.

Pascal es recursivo y gran parte del trabajo destinado a establecer la recursividad y llamados cruzados entre rutinas, puede ser dejado al lenguaje objeto.

En contraposición, todas las declaraciones que se generen al programa en PASCAL (como las declaraciones de archivos o etiquetas) deberán aparecer antes de las proposiciones y ésto puede representar una desventaja. Por ejemplo, las etiquetas que aparezcan en el programa fuente serán emuladas por otras etiquetas en el programa objeto, pero para saber el número exacto de ellas que se emplearán, se requiere revisar por completo al bloque respectivo del programa fuente.

Al generar FORTRAN el código consiste de proposiciones muy sencillas, que lo asemejan a un lenguaje de ensamblador, pero donde no se tiene que controlar a la máquina objeto. Esto último, sin embargo, no es una gran ventaja, pues implica tener que hacer el trabajo de generar lenguaje ensamblador sin ganar las ventajas del mismo, como es el poder manejar la memoria y los recursos de la computadora sin las restricciones que impone un lenguaje de programación como FORTRAN.

12) Hay una versión de PASCAL UCSD para las computadoras PC. Dicha versión abarca completamente a la versión de APPLE. Esto implica ganancia en memoria principal y ganancia en memoria secundaria, tanto por la mayor capacidad en los discos flexibles, como por la posible disponibilidad de un disco duro.

13) Por último, no se escogió otro lenguaje más apto para escribir un compilador, como lo es C, debido a que cuando se hizo la selección del lenguaje anfitrión y del lenguaje objeto no se encontró una versión para APPLE que contemplara reales y entonces se habría tenido que diseñar un cuerpo de rutinas para almacenar y operar números reales, lo cual podría crear ineficiencias en el código y retrasaría la implantación del resto del lenguaje.

III. ESTRUCTURA DEL COMPILADOR

La estructura que actualmente presenta el compilador es la misma que tenía en la versión antigua. A continuación se describe brevemente las fases que constituyen al compilador, junto con los programas de utilería ya mencionados. Todos los programas están escritos en PASCAL.

A) ANALIZADOR LEXICOGRAFICO Y SINTACTICO

Fase I de Compilación.

El primer analizador recibe al programa fuente, frase, por frase, produciendo una sucesión de átomos, llamada "texto limpio"; esta información es tomada por el siguiente analizador generando el árbol sintáctico que representa a la frase dada. Simultáneamente se va construyendo el diccionario que comprende a identificadores, cadenas y números.

Entrada:

1) Programa Fuente.
2) Tabla de enteros que representa a la gramática.
3) Diccionario que comprende a los nombres de todos los símbolos no-terminales que hay en la gramática. También se

denomina como diccionario de variables metalingüísticas.

4) Diccionario que contiene a todas las palabras clave en el lenguaje. Se utiliza para saber cuando una cadena alfabética representa a un identificador o a una palabra clave.

Salida:

1) Listado del programa fuente, junto con los errores encontrados y números de frase.

2) Arboles sintácticos.

3) Diccionario de los identificadores, números y cadenas aparecidos en el programa fuente.

Se puede apreciar un diagrama que describe al analizador en la figura 5.1 .

B) ANALIZADOR SEMANTICO Y GENERACION DE CODIGO

Fase II de Compilación

Por cada árbol sintáctico se aplica el análisis semántico y se hacen las operaciones correspondientes; se emiten los errores encontrados, indicando el número de frase a que pertenecen, y finalmente se graba el código consecuente.

Entrada:

1) Arboles sintácticos.

2) Diccionario de identificadores, números y cadenas.

Salidas:

1) Listado de errores.

2) Código. (Programa en PASCAL).

La figura 6.1 muestra un diagrama que representa al analizador.

C) PROGRAMAS DE UTILERIA

1) Generador de la Tabla Gramatical.

Toma la gramática del lenguaje escrita en notación BNF

y produce una tabla de enteros que representa la gramática y un archivo con los nombres de las variables metalingüísticas en la gramática. (Ver la figura 4.1).

2) Generador del Diccionario de Palabras Clave
Toma un archivo que contiene la lista de palabras clave en el lenguaje y produce un par de archivos que conformarán la parte de la tabla de símbolos correspondiente a palabras claves. (Ver la figura 4.2).

En los próximos capítulos se presentarán cada uno de los reconocedores y programas de utilería mencionados, describiéndolos en detalle y resaltando las modificaciones que tienen respecto a la versión original.

Comenzaremos con la revisión de los programas de utilería en el capítulo siguiente.

IV. PROGRAMAS DE UTILERIA

Como ya se mencionó en el capítulo anterior, los programas de utilería consisten en dos procedimientos que generan un conjunto de cuatro tablas que serán utilizadas por el compilador para inicializar el diccionario de átomos (que forma parte de la Tabla de Símbolos) y para llevar a cabo los reconocimientos lexicográfico y sintáctico.

El primer programa se encarga de procesar la gramática del lenguaje estadístico, que se encuentra escrita en BNF y genera un par de tablas gramaticales que serán empleadas por el reconocedor sintáctico.

El segundo programa de utilería crea a partir de un archivo con las palabras clave del lenguaje estadístico, un par de tablas que conforman un diccionario de palabras clave que inicializará a la Tabla de Símbolos del compilador.

En general se mantuvo la estructura de los dos programas de utilería que tenían en la versión de la NOVA e inclusive se mantuvieron los nombres de los programas: HT7 para el procesa-

dor de la gramática y HPC2 para el generador del diccionario de palabras clave.

El diseño original de los programas se puede encontrar en García [G1].

Las modificaciones a los programas residen principalmente en el tipo de archivos generados. Anteriormente se generaban archivos formados por caracteres ASCII, que cumplían los requerimientos para leerse en FORTRAN, mediante el formato libre. Ello implicaba un gran desperdicio de espacio en caracteres blancos y comas. En la nueva versión se generan registros de enteros y de caracteres, que disminuyen por mucho el espacio utilizado, y el tiempo dispuesto en grabar y leer toda esa información.

Por ejemplo la tabla gramatical ocupa en PASCAL cerca de 3K, contra 55K que disponía en la NOVA y la lectura ahora tarda cerca de 15 segundos, en vez de los 5 minutos que anteriormente se requerían.

A) GENERADOR DE LA TABLA GRAMATICAL (HT7)

Toma una gramática escrita en BNF (GTO.TEXT), que describe a la sintaxis del lenguaje y genera dos archivos: una tabla de enteros (GRAMGEN) y una sucesión de caracteres (METASIMB). (Ver el diagrama).

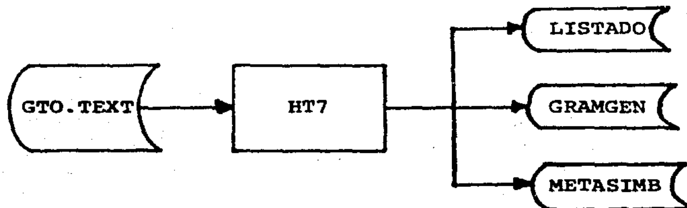


Figura 4.1 Diagrama del programa HT7

El archivo "METASIMB" contiene los nombres de los símbolos no-terminales en la gramática. Se emplea durante el análisis sintáctico, al imprimir el contenido del árbol sintáctico. Esta tabla, por cierto, no se emitía en el diseño original, sino que es una inovación de la versión para APPLE.

La tabla "GRAMGEN" representa a la gramática y es tomada para efectuar el análisis sintáctico.

Este diseño que se ha dado al analizador sintáctico tiene la gran ventaja de que si se desea modificar la sintaxis del lenguaje, basta con corregir el archivo GTO.TEXT, para describir al nuevo lenguaje, correr el programa HT7 y por último, sustituir los archivos GRAMGEN y METASIMB por las nuevas versiones.

La tabla GRAMGEN* está diseñada para un reconocedor de arriba hacia abajo.

El archivo GTO.TEXT contiene la gramática del lenguaje que está escrita con las convenciones de BNF**.

Cabe mencionar que una de las principales características de la gramática es que se permiten dos tipos de producciones: Por un lado, las producciones que llamo normales o tipo-P, que representan la derivación de un símbolo no-terminal en una sucesión de símbolos gramaticales; la otra clase de producciones, de tipo-B, consisten de símbolos no-terminales que derivan un número entero. Estas últimas representan llamados a rutinas semánticas y se incluyen dentro de los lados derechos de las

---

* Una descripción a fondo de la estructura de la tabla gramatical y de como es utilizada por el analizador sintáctico, se encuentra en los trabajos de Luis Legarreta [L1] y de Javier García [G1].

** Las convenciones utilizadas para escribir la gramática, así como a la gramática misma, se encuentran en el apéndice A.

producciones tipo-P. Es mediante estos símbolos que primordialmente se dirige el reconocedor semántico.

El "LISTADO" es un archivo que puede dirigirse a cualquier dispositivo de salida (pantalla, impresora, disco) y muestra la gramática leída, la tabla gramatical producida y el diccionario de símbolos no-terminales.

B) GENERADOR DEL DICCIONARIO DE PALABRAS CLAVE (HPC2)

A partir de la lista de las palabras clave del lenguaje (PALCL7.TEXT), se construyen dos tablas que conforman un diccionario que posteriormente será tomado por el compilador para reconocer y formar átomos y para llevar a cabo el reconocimiento sintáctico. Las tablas están en los archivos PCVDT y PCCHAR.

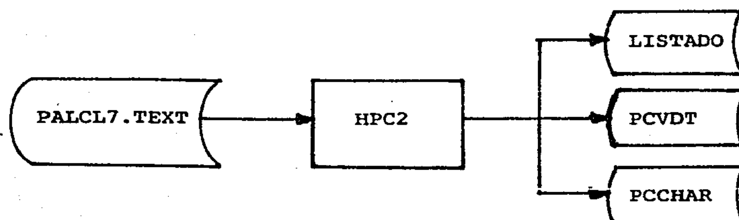


Figura 4.2 Diagrama del programa HPC2

El archivo PALCL7 contiene las palabras clave separadas por blancos y por caracteres de fin de línea. Este texto fué escrito, como en el caso de la gramática, con el editor de PASCAL[A2].

El diccionario se compone de dos partes; primero, la sucesión de caracteres que componen cada palabra, junto con la longitud de la palabra (PCCHAR). La otra tabla (PCVDT), consiste de

apuntadores a PCCHAR, hacia los caracteres en donde comienza cada una de las palabras. En el siguiente capítulo se podrá observar más a fondo como PCCHAR y PCVDT son empleadas en el compilador.

Este diccionario formará parte de la tabla de símbolos del compilador y se empleará para reconocer a las palabras clave en el programa fuente.

El "LISTADO" es un reporte que muestra las palabras clave junto con sus respectivas longitudes; el número de caracter dentro de CHARS, en donde comienza cada palabra y que son los valores que contienen PCVDT y la posición que cada palabra tiene dentro de PCVDT.

V. ANALIZADOR LEXICOGRAFICO Y SINTACTICO

A) INTRODUCCION

Esta primera fase en la compilación toma al programa fuente y entrega un listado del mismo numerado por cada frase sintáctica y con los errores que contenga. Entrega también, un diccionario de los identificadores, números y cadenas en el programa, que más tarde será cargado a la tabla de símbolos y los árboles sintácticos de todas las frases aceptadas.

El procedimiento detallado se exhibe y explica a continuación:

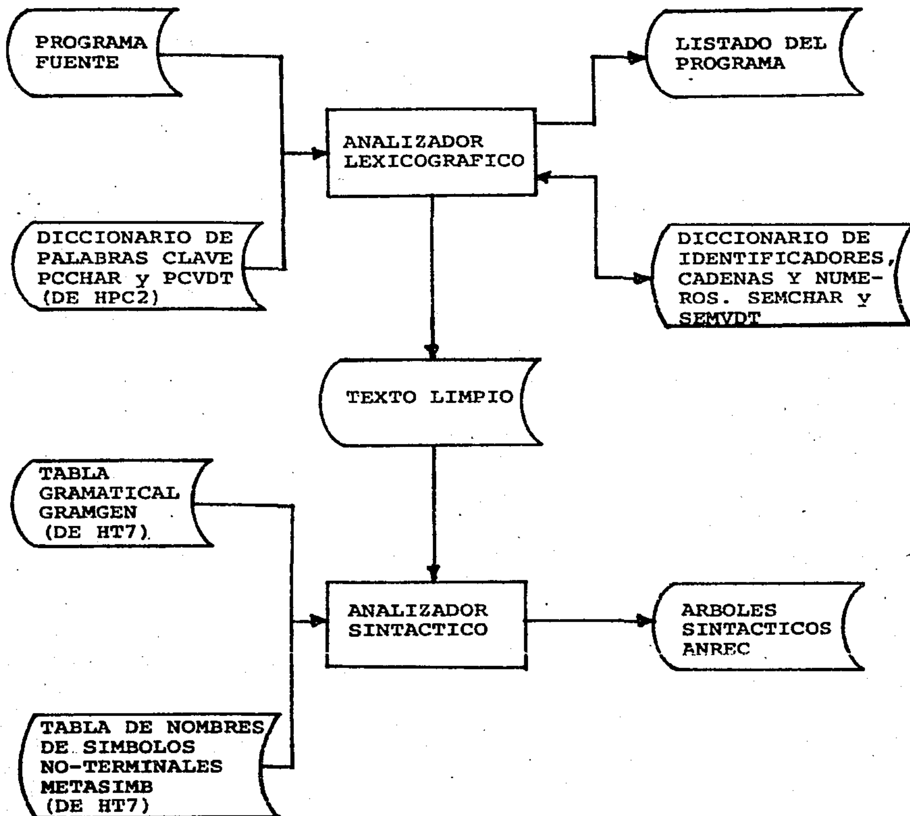


Figura 5.1 Diagrama del analizador lexicográfico-sintáctico.

El control de la compilación la tiene el analizador lexicográfico, que al hallar el fin de cada frase, llama al reconocedor sintáctico y después regresa a procesar la siguiente frase, hasta acabar con el programa fuente.

Previo a la lectura del programa fuente, el analizador lexicográfico carga el diccionario de palabras clave, el cual le sirve para diferenciarlas de los identificadores. También el analizador sintáctico carga algunas cosas al inicio de la ejecución: la tabla gramatical con la que puede reconocer o rechazar las frases y la lista de los nombres de los símbolos no-terminales de la gramática, que usará para imprimir los árboles sintácticos en forma legible.

La rutina léxica va leyendo, frase por frase, el programa fuente y determina cuando llega al final de cada frase (al encontrar un punto y coma o la palabra clave INICIO). El reconocedor va limpiando la versión fuente y generando átomos, los cuales va colocando en el texto limpio; así mismo va agregando al diccionario de átomos (que se inicializó con las palabras clave), los identificadores, números y cadenas encontradas (este nuevo diccionario es grabado en SEMCHARS y en SEMVDT); también va emitiendo un reporte de lo que va leyendo, con los errores que encuentra (por ejemplo, símbolos ilegales, números mal construidos, etc). El reporte está numerado por frase sintáctico y no por línea. Es decir, una frase abarca hasta un punto y coma o hasta la palabra clave INICIO, siempre y cuando ésta no sea el primer elemento de la frase.

Veamos un ejemplo tomando el siguiente programa:

```
              % UTILIZACION  DE EXPRESIONES;
$$D+
INICIO
   CONSTANTE
      TAMEDADES = 100;
   ARREGLO
      EDADES, SEGEDADES [ 1:2, 1:TAMEDADES ];
   REAL
      ED, POBTOTAL, EDADMUJERES;

   % INICIALIZACION DEL ARREGLO EDADES ;
   EDADES:= [[ 2, TAMEDADES : 9 ( 1, 7, 3 ( PI, E, 13 ), 11 ) ]];

   % SUMA DE LAS EDADES DE LAS MUJERES ENTRE 5 Y 90 A&OS ;
   EDADMUJERES:=(( (EDADES[1, ]>=5) Y (EDADES[1, ]<=90) )*EDADES[1, ])[.];

   % INICIALIZACION DE SEGEDADES ;
   SEGEDADES[ 2, 40_90 & 1_10 ]:=
      [[ TAMEDADES : EDADES[ 1, 20&10_20&10 ] ]];

FIN;
```

Figura 5.2 Ejemplo de un programa hecho en el Lenguaje Estadístico.

Para este programa se emite el siguiente reporte.

```
(    1)              % UTILIZACION  DE EXPRESIONES;
(    1) $$D+
(    2) INICIO
(    2)    CONSTANTE
(    2)       TAMEDADES = 100;
(    3)    ARREGLO
(    3)       EDADES, SEGEDADES [ 1:2, 1:TAMEDADES ];
(    4)    REAL
(    4)       ED, POBTOTAL, EDADMUJERES;
(    5)
(    5)    % INICIALIZACION DEL ARREGLO EDADES ;
(    5)    EDADES:= [[ 2, TAMEDADES : 9 ( 1, 7, 3 ( PI, E, 13 ), 11 ) ]];
(    6)
(    6)    % SUMA DE LAS EDADES DE LAS MUJERES ENTRE 5 Y 90 A&OS ;
(    6)    EDADMUJERES:=(( (EDADES[1, ]>=5) Y (EDADES[1, ]<=90) )*EDADES[1, ])[.]
;
(    7)
(    7)    % INICIALIZACION DE SEGEDADES ;
(    7)    SEGEDADES[ 2, 40_90 & 1_10 ]:=
(    7)       [[ TAMEDADES : EDADES[ 1, 20&10_20&10 ] ]];
(    8)
(    8) FIN;
(    9)
(    9)
(    9)
  **** ACABO EL ANALIZADOR LEXICO-SINTACTICO ****
         0 ERRORES
```

Figura 5.3 Reporte que emite el analizador lexicográfico para el programa de la figura 5.2

Al encontrarse el fin de frase, la rutina léxica llama al reco nocedor sintáctico, el cual por medio de la tabla gramatical analiza al texto limpio.

Si el análisis resulta correcto, se graba el árbol sintáctico y opcionalmente se imprime en el reporte. En caso contrario se emite al reporte el error respectivo y no se graba el árbol sintáctico correspondiente, ya que el error imposibilita su construcción.

En general, la estructura de este programa sigue siendo la misma que en la versión de la NOVA. Sólo se hicieron algunos cambios y mejoras que más tarde se detallarán.

B) ANALIZADOR LEXICOGRAFICO

Como ya se había mencionado antes, la tarea del analizador lexi cográfico consiste en reconocer los átomos ("tokens") que constituyen al programa fuente y convertirlos a una notación interna llamada texto limpio; además elimina los comentarios y los espacios superfluos y también detecta las opciones de compilación, llevando a cabo las acciones pertinentes.

De acuerdo al primer carácter de una cadena, se determina el tipo de átomo que se tiene y entonces, se pasa a ejecutar la rutina respectiva.

Cuando el átomo corresponde a una palabra clave, identificador, número o cadena, se codifica en una pareja de enteros. El pri mer elemento indica el tipo de átomo y el otro integrante es un apuntador al diccionario de átomos.

Los valores que representan al tipo de átomo son:

2000 - palabra clave
1020 - identificador
1010 - número
1040 - cadena de caracteres

El diccionario está constituido por dos arreglos: CHARS, que es un arreglo de caracteres donde viven las cadenas que constituyen a los átomos y VARDIC, que es un arreglo de enteros que apuntan a aquel.

Así, por ejemplo, VARDIC [i] apunta al lugar de CHARS, donde está la cadena del átomo representado por VARDIC [i]. Supongamos que dicho apuntador es j, en CHARS [j] existe la longitud de la cadena escrita en binario, sea ℓ. Desde CHARS [j+1] hasta CHARS [j+ℓ] están los caracteres que constituyen al átomo. Esta estructura se ilustra en la siguiente figura:

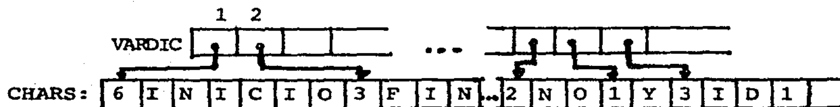


Figura 5.4 Diccionario de átomos.

Para las palabras clave e identificadores, los átomos en el texto limpio apuntan al arreglo VARDIC y para los números y cadenas se apunta directamente a CHARS.

La forma en que es administrado el diccionario es secuencial, es decir, los métodos de búsqueda e inserción son lineales. Se decidió no implantar una función de dispersión porque en esta etapa de desarrollo del compilador, la optimización del sistema no está dentro de los principales objetivos a lograr. La implantación de algoritmos más eficientes en el manejo de la tabla de símbolos se deja como mejora futura.

Al comienzo del programa, el diccionario es incializado con la información referente a palabras clave*, guardándose la primera localidad libre de VARDIC en la variable IVDTKW.

---

* Como ya se ha mencionado, esta información se obtiene a partir de los archivos PCCHAR y PCVDT generados por el programa de utilería HPC2.

Al tomarse un identificador, este se rastrea a lo largo del diccionario, hasta encontrarlo o darlo de alta. Mediante el valor de IVDTKW se determina si el nombre corresponde a un identificador o a una palabra clave.

Los números son revisados sintácticamente y luego se almacenan íntegramente, incluyendo el posible signo inicial.

Los textos son almacenados comenzando con unas comillas ", para señalar que es una cadena.

Si los átomos corresponden a signos de puntuación, separadores, operadores o cualquier caracter especial, se almacenan en el texto limpio con el valor decimal que tienen en el código ASCII. De esta manera no hay confusión con los otros átomos.

Al hallarse un punto y coma, el texto limpio es tomado por el reconocedor sintáctico y luego se continúa con la construcción del texto limpio de otra frase o se finaliza la ejecución del programa.

El texto limpio queda almacenado en el arreglo TXT.

C) ANALIZADOR SINTACTICO

1). Presentación y justificación

El papel del analizador sintáctico consiste en tomar la sucesión de átomos que constituyen al texto limpio y revisar que cumplan con las reglas sintácticas del lenguaje.

En caso de no hallar errores, el árbol derivado se graba en el archivo ANREC y en caso contrario, se emite un error al listado del programa.

El analizador efectúa un reconocimiento de arriba hacia abajo

("top-down"), siguiendo el método de descenso recursivo ("recursive descent") y valiéndose de la técnica de prueba y error ("backtrack").

Se escogió este analizador porque en este momento presenta una serie de ventajas sobre otros tipos de reconocedores, como se explica a continuación.

- Los algoritmos para generar la tabla gramatical y para hacer el reconocimiento son sencillos y fáciles de implantar. Cabe mencionarse que la misma técnica empleada en esta versión fué la utilizada en la máquina NOVA, en el lenguaje FORTRAN.

- Por otra parte en los analizadores de abajo-hacia-arriba LR, resulta muy tedioso elaborar la tabla gramatical y si no se dispone de un generador de tablas no se recomienda su elección. Veamos que dicen al respecto Aho y Ullman ([A3], pág. 197).

> "La principal desventaja del método, es que es demasiado trabajo el implantar un reconocedor LR a mano, para una gramática típica de un lenguaje de programación. Se necesita una herramienta especializada -un generador de reconocedores sintácticos LR".

No se escogió un reconocedor LL porque también se requiere un generador de tablas gramaticales y porque imponen muchas restricciones a las gramáticas.

Se descartó también un algoritmo ad-hoc, porque es muy tedioso elaborarlo y cuando la gramática es compleja, se vuelve confuso y difícil de modificar.

Finalmente se rechazaron otras técnicas de desplazamiento-reducción (shift-reduce) como la de precedencia de operadores porque el lenguaje no vive alrededor de operadores y

porque resultan tan difíciles de instalar como los LR.

- El reconocedor elegido abarca un gran número de gramáticas libres de contexto y sólo impone algunas condiciones:

  Que la gramática carezca de símbolos muertos (que todos los símbolos generen al menos una cadena de únicamente símbolos terminales), que carezca de símbolos inalcanzables (que cualquier símbolo pueda ser derivado desde el símbolo distinguido) y que carezca de ciclos (que un símbolo no pueda derivar en uno o más pasos, únicamente a él mismo). [H1].

  También hay otros requisitos, pero que sólo se involucran en la presentación de la gramática:

  Todas las producciones que se deriven del mismo símbolo deben presentarse juntas. Si tengo dos producciones, en la que una es prefijo de otra, la más grande debe aparecer primero. Como la producción vacía es un caso especial de lo anterior, debe ir al último.

  Pasemos ahora, a ver las restricciones de un lenguaje LL(1). Si se tienen dos producciones definidas por un mismo símbolo entonces:

a) No debe ocurrir que algún símbolo terminal con el que comience alguna derivación de una de las producciones, sea también el símbolo que inicie alguna derivación de la otra producción.

b) A lo más una producción puede derivar a la palabra vacía.

c) Si alguna producción genera a la palabra vacía, ningún símbolo terminal que aparezca inmediatamente a la derecha de esa producción en cualquier derivación de la gramática, pue de ser el mismo símbolo que inicie alguna derivación de la otra producción.

Como puede observarse son muchas las gramáticas que no son LL(1), incluyendo la de este compilador. (Se puede ver la gramática en el Apéndice A).

Algunos métodos para elaborar reconocedores LR, como el método simple SLR fallan en gramáticas que otros métodos triunfan. ([A3], pág. 198). Además si la gramática es ambigua, entonces no puede ser LR. (Ver el teorema respectivo en Harrison [H2], pág. 505, teorema 13.2.1).

- Modificar la sintaxis del lenguaje y adaptar el reconocedor a la nueva versión es un proceso sencillo y rápido pues basta con modificar el archivo que contiene a la gramática por medio de un editor y correr el generador de la tabla gramatical.

Sin embargo, para ser justos, debo mencionar los inconvenientes de este reconocedor.

La principal desventaja es que es muy lento por la técnica de prueba y error ("back-track"). Provoca que el tiempo de análisis dependa en forma más que lineal de la longitud de la cadena.

Finalmente, como en este momento de desarrollo del compilador se busca la facilidad de estar modificando la gramática sin hacer mucho esfuerzo, se escogió la opción presentada.

Una descripción a fondo de los algoritmos de generación de la tabla gramatical y del reconocimiento sintáctico se encuentra en Legarreta [L1].

El programa que genera la tabla gramatical es el HT7, que ya se discutió con anterioridad. La rutina que hace el reconocimiento se llama PARSER y está dentro del analizador lexicosintáctico.

2). Características principales

La gramática permite dos tipos de producciones: Las tipo - B y las tipo - P.

Las producciones tipo P definen un elemento no-terminal en una serie de alternativas, cada una formada por una sucesión de símbolos terminales y no-terminales escritos en notación BNF.

Las producciones tipo B definen un símbolo no-terminal en un número entero. De acuerdo al valor o al rango donde esté dicho número, el símbolo tiene uno de los siguientes significados:

a) 2000 < número < 4000

El átomo esperado es una palabra clave. Para determinar cual palabra es, al número se le resta el valor de 2000 y el resultado es un índice al arreglo VARDIC, que forma parte del diccionario. El entero 2000 y el número resultante, son el par de valores que son depositados en el texto limpio.

b)

Valor	Tipo de átomo
1020	identificador
1010	número
1040	texto

En estos casos el texto limpio debe contener la pareja cuyo primer elemento es el tipo del átomo y el otro elemento es un apuntador al diccionario. Si no hay error, se construye un nodo que contenga ese apuntador y se intitula con el tipo del átomo.

c) 4000 < = número < 10000

El símbolo representa un llamado semántico. Esto significa que a cada rutina o segmento de código en el analizador semántico, corresponde un valor en este rango.

Se construye un nodo para el árbol, con el valor del símbolo.

Este tipo de símbolos es de gran importancia, pues constituye el esqueleto que dirige y controla al reconocedor semántico. Cualquier acción realizada por éste es decidida y dirigida a partir de dichos llamados.

Este diseño que presenta el código semántico representa una gran ayuda para ir construyendo el compilador. Se evita tener que seguir los árboles sintácticos, nodo por nodo, lo cual es un seguimiento complicado y tedioso que además vuelve confuso y obscuro al programa. Además cualquier modificación a la gramática acarrea cambios forzosos a la estructura del programa que conforme va creciendo se hacen más difíciles de implantar.

Con el diseño de llamados en el árbol, basta con colocar dichos elementos en puntos estratégicos de la gramática, para que la semántica decida cuales rutinas ejecutar y en que momentos.

Como cada rutina sólo tiene que revisar un grupo de nodos y no a todo el árbol sintáctico, las modificaciones al analizador semántico ocasionadas por cambios gramaticales son más fáciles de efectuar que en el caso de un programa que analice al árbol nodo por nodo.

El árbol generado del seguimiento de la gramática consiste de dos arreglos lineales. El primer arreglo (A), contiene el número de alternativa tomado para el símbolo no-terminal representado o un apuntador al diccionario o un llamado semántico. El otro arreglo (NP), contiene índices al primer nodo hermano a la derecha o en su defecto, al nodo ascendiente derecho más próximo.

Además hay un arreglo adicional poco importante (PN), que contiene apuntadores a la tabla de nombres de los símbolos no-terminales de la gramática (DMS) y se utiliza para que el reporte del árbol sintáctico sea más legible.

A continuación se presenta un ejemplo de una línea de programa fuente con el texto limpio y el árbol sintáctico. El árbol consiste de cuatro líneas: números de nodo (línea I), nombres de los símbolos no-terminales representados (línea II), números de alternativa (línea III) y apuntadores a otros nodos (línea IV).

```
2)
2)      REAL X, YY, I, FINARCHIVO ;

2000       4    1020      47      44    1020      48      44    1020
  49      44    1020      50      59


   1       2       3       4       5       6       7       8       9
PRINC   INIDV   P       DECL    *SEM*   L DE    1020    RDS     1020
   1       2       1       2    4011       1      47       1      48
  15       3      15      15       6      15       8      15      10

  10      11      12      13      14
RDS     1020    RDS     1020    RDS
   1      49       1      50       2
  15      12      15      14      15
```

Figura 5.5 Segmento del listado emitido por el reconocedor lexicográfico mostrando el texto limpio y el árbol sintáctico.

La línea del programa fuente primero se pasa al texto limpio como sigue:

La palabra clave REAL se codifica en la pareja (2000,4). El número 2004 es el valor con que se define la producción B(REAL) en la gramática.

Los identificadores se codifican también en parejas: (1020,n), donde n apunta al diccionario de identificadores, a donde se define dicho apuntador. Ya dentro del árbol los identificadores aparecen bajo el nombre de "1020", como en los nodos número 7, 9, 11 y 13. El número de alternativa de dichos nodos es precisamente el apuntador al diccionario de identificadores.

Los otros componentes del texto limpio representan a los signos de puntuación y se constituyen a partir del valor en decimal que dichos símbolos tienen en el código ASCII. En el ejemplo presentado sólo hay dos tipos de signos de puntuación y son las comas (los elementos 44 en el texto limpio) y el punto y coma (el elemento 59.

Dentro del árbol sintáctico puede observarse un llamado semántico en el nodo 5 intitulado "*SEM*" y contiene el llamado 4011,

que se encarga de declarar a las tres variables.

D) CAMBIOS EFECTUADOS AL ANALIZADOR LEXICOGRAFICO Y SINTACTICO

i) Se redujo considerablemente el espacio utilizado por los archivos de entrada y salida: GRAMGEN, METASIMB, PCCHAR, PCVDT, SEMCHARS, SEMVDT y ANREC, así como el tiempo necesario para la lectura y grabación de esos archivos.

ii) Dispositivos para el programa fuente y para el listado del programa.

La fuente de donde se tome el programa a compilar, ahora puede ser cualquier archivo en disco y también puede ser la consola, creando el programa al momento de compilar.

El dispositivo a donde se dirija el listado, ahora puede ser la pantalla, la impresora o cualquier archivo en disco.

iii) Modificación a la impresión del árbol sintáctico.

Por cada nodo del árbol se imprimen cuatro cosas: número de nodo; nombre del nodo (en la versión de la NOVA no aparecía); número de alternativa, apuntador al diccionario o llamado semántico y apuntador al nodo hermano o ascendiente más próximo.

iv) Almacenamiento de número en el árbol sintáctico.

En la versión de la NOVA, el nodo que representaba un número, apuntaba hacia el arreglo VARDIC. En la nueva versión, el nodo señala directamente al arreglo CHARS a fin de ahorrar tiempo al leerlo.

v) Modificaciones al analizador sintáctico y a la gramática.

El algoritmo para hacer el reconocimiento sintáctico es recursivo, por lo que en la versión de la NOVA se tuvo que simular la recursividad. En la versión actual ya no se tuvieron que hacer alteraciones.

Se mejoró el análisis de identificadores, de número y de cadenas. Anteriormente, para definir a estos átomos en la gramática se tenía que recurrir a un símbolo no-terminal adicional. Por ejemplo, para definir al identificador se tenían las producciones: P(identif) = (ident) y B(ident) = 1020. Ahora basta con tener una definición directa: B(identif)=1020 y el programa se entera de que hay que crear un nodo. Estos cambios también se aplicaron a las definiciones de texto y de número.

Se adicionaron producciones que representan llamados semánticos y que se han discutido anteriormente. El programa se encarga de detectar esos símbolos y de incluirlos en el árbol. La ventaja de hacer estas definiciones en forma directa, es que la gramática es menos compleja y el espacio ocupado por la tabla gramatical disminuye bastante.

Aparte de los cambios en la gramática que acabo de mencionar, hay otros que conciernen directamente a la sintaxis y que expondré a continuación.*

a) Definición de arreglos explícitos. Al declarar las dimensiones de un arreglo explícito se permitían solamente números, ahora se permiten números y constantes.

b) Antes, si una frase comenzaba con la palabra INICIO, el árbol derivado tomaba la parte correspondiente a proposiciones (independientemente de que fuera ésto o una declaración) y hasta que llegaba a la definición de proposición incondicional, tomaba la alternativa: INICIO (P), donde (P) era prácticamente el símbolo distinguido, y podía ya el reconocedor tomar la parte de declaraciones. Para mayor claridad exhibiré la gramática anterior:

---

* La descripción de la gramática tal y como era anteriormente, se encuentra en García G1 .

```
<PRINCIPIO> :: = <P> ";"
<P> ::=<DECL> |
      <L.ETIQ.> <PROP.GRAL.>
<PROP.GRAL.> :: = ...|
      <P.INC.>
<P.INC.> :: = <PR.INC.> <TERMINA>
<PR.INC.> :: = ...|
       "INICIO" <P>|...
```

Como puede verse si una frase comenzaba con INICIO no podía dicidirse con los primeros nodos (en particular con el nodo <P>), si la frase correspondía o no, a una proposición.

Se hicieron algunas modificaciones y la gramática quedó como sigue:

```
<PRINCIPIO> :: = <INICIO o VACIO> <P> ";"
<INICIO o VACIO> :: = "INICIO" |e
<P> :: = <DECL.> |
         <L.ETIQ.> <PROP.GRAL.>
<PROP.GRAL.> :: = ...|
         <P.INC.>
<P.INC.> :: = <PR.INC.> <TERMINA>
<PR.INC.> :: = ...|
          "INICIO" |
           ...
```

Ahora el símbolo <P> nunca aparece en el lado derecho de alguna producción y se puede saber de inmediato si se tiene una declaración o no.

Cuando se tiene una proposición compuesta, ésta se rompe artificialmente en dos frases, exactamente al terminar la palabra "INICIO". A continuación expongo esta situación.

c) Una frase, tanto para el reconocedor léxico, como para el sintáctico, abarca hasta hallar un punto y coma. Esta propiedad implica que la gramática se vuelve muy compleja.

Cuando se tiene un grupo de proposiciones sencillas o compuestas que están anidadas, en la gramática se tiene que hacer una serie de llamados recursivos a los símbolos que definen dichas proposiciones y entonces los árboles crecen desmesuradamente, posiblemente hasta llegar a consumir todo el espacio disponible.

Debido a ello, el reconocedor léxico se alteró para que cada vez que encuentre la palabra "INICIO" (siempre y cuando no sea la primera palabra de la frase) inserte un punto y coma al texto limpio y llame al reconocedor sintáctico, el cual de hecho procesa una producción completa que termina en "INICIO". De esta manera, la frase realmente se divide a nivel sintáctico. La proposición compuesta se vuelve una especie de "encabezado" seguido de las proposiciones elementales que la componen.

Las producciones modificadas son:

```
        <PR.INC.> :: = "INICIO" <P>
por     <PR.INC.> :: = "INICIO"
        <CUERPO ITERACION> :: = "INICIO" <L.ETIQ.> <PROP.GRAL.>
por     <CUERPO ITERACION> :: = "INICIO"

        <SELECCION> :: = "LA" <EXP.> "DE INICIO" <L.ETIQ.>
                         <PROP.GRAL.>
por     <SELECCION> :: = "LA" <EXP.> "DE INICIO"

        <DECL.> :: = "RUTINA" <Z TIPO>  <IDENTIF.> <ZONA>
                          DE PARAMETROS> "INICIO" <P>
por     <DECL.> :: = "RUTINA" <Z TIPO> <IDENTIF.> <ZONA
                          DE PARAMETROS> "INICIO"
```

Con ésto, la gramática se volvió menos compleja, de menor tamaño y los árboles sintácticos generados son acotados.

d) La parte izquierda de una asignación consistía de un identificador sucedido de un campo o de una selección de arreglo

o de nada. Ahora permite un campo opcional y una selección de arreglo opcional:

antes:

```
<ASIG.> :: = <EST.REF.> ":="...
<EST.REF.> :: = <IDENTIF.> <OPER.ESTRUC.>
<OPER.ESTRUC.> :: = "." <OP.EST.> <EXP.SELEC.>
<EXP.SELEC.> :: = <EXP.SELEC. DE ARR.>|e
```

ahora:

```
<ASIG.> :: = <EST.REF.> ":="...
<EST.REF.> :: = <IDENTIF.> <OPER.ESTRUC.> <EXP.SELEC.>
<OPER.ESTRUC.> :: = "." <OP.EST.>|e
<EXP.SELEC.> :: = <EXP.SELEC. DE ARR.>|e
```

vi) Incorporación de opciones de compilación. Hasta el momento se han introducido tres opciones.

La sintaxis es como sigue:

```
<opción de compilación> :: = "$$" <letra> <señal>
<letra> :: = "A" | "C" | "D"
<señal> :: = "+" | "-"
```

Los signos + y - activan o desactivan la opción, respectivamente.

La línea debe estar fuera de todo comentario y puede aparecer en cualquier lugar de la línea.

1) $$C

Impresión del texto limpio.
Por cada frase se imprime la sucesión de números enteros que componen al texto limpio.
Por omisión, la opción está desactivada.

2) $$A

Impresión del árbol sintáctico.
Por cada frase se imprime el contenido del árbol sintáctico.
Por omisión, la opción está desactivada.

3) $$D

Monitoreo del análisis semántico.

Por cada frase se va reportando la actividad del reconocedor semántico, junto con los errores encontrados.

Por omisión esta opción está desactivada.

Dado que la semántica constituye otro programa, esta opción se pasa a través del árbol sintáctico. En el primer nodo se pone un número de alternativa negativo, que indica que es una opción de compilación. En el segundo nodo se coloca el código de operación (activar o desactivar). Así, no hay confusión con un árbol producido a partir de un seguimiento sintáctico.

E) PRINCIPALES ESTRUCTURAS DE DATOS Y RUTINAS DEL RECONOCEDOR

Los nombres de archivos y arreglos aparecen en mayúsculas, los de escalares en minúsculas.

Estructuras de Datos.

Archivos:

Nombre Lógico	Nombre Físico	Función
FTE	Los proporciona	Programa Fuente
LST	el usuario	Listado de compilación
RESUNO	PCCHAR y SEMCHARS*	Diccionario de
RESDOS	PCVDT y SEMVDT*	átomos
GRAM	GRAMGEN	Tabla gramatical
METAS	METASIMB	Tabla con nombres de símbolos no-terminales
ANSIN	ANREC	Arboles sintácticos

* El primer archivo es de entrada, el otro de salida.

Arreglos y Escalares:

VARDIC	Parte del diccionario de átomos. Contiene apuntadores al arreglo de caracteres.
ivdt,ivdtkw	Apuntadores a VARDIC. Ivdtkw apunta a la primera palabra que no es clave.
CHARS	El arreglo del diccionario que contiene las cadenas que constituyen a los átomos.
icht	Apuntador a CHARS.
PS	Contiene la tabla gramatical.
DMS	Diccionario de nombres de símbolos no-terminales.
mando y mingram	Símbolo distinguido de la gramática.
A, NP, PN	Contienen al árbol sintáctico. Respectivamente representan al número de alternativa, al apuntador al nodo hermano o ascendiente y al nombre del nodo (índice a DMS).
TXT	Arreglo donde se almacena al texto limpio.
ixp	Apuntador a TXT.
CONV	Contiene los valores que denotan la clase de caracter a que corresponda un símbolo de la entrada.
nlinea	No. de frase sintáctica en proceso.
faults	Contador de errores en el programa fuente.
s	Caracter actualmente leido, o entregado.
nsbuf	Siguiente caracter a entregar.

Rutinas:

ERROR(N)	Procesa al error cuyo código es N.
LEEGRAMATICA	Carga la gramática y el diccionario de nombres a los arreglos PS y DMS.
INIDIC	Carga el diccionario de palabras clave inicializando a los arreglos CHARS y VARDIC.
INICON	Asigna valores al arreglo CONV.
INICIAL	Llama a las tres rutinas anteriores, abre los archivos para el programa fuente, para

	el listado y para los árboles; inicializa apuntadores y otras variables escalares.
FINLEXICO	Cierra archivos de salida (LST, ANSIN) y graba el diccionario de átomos (SEMCHARDS y SEMVDT).
LEESIM(S)	Entrega en S al símbolo actual y lee al siguiente símbolo que entregará (nsbuf).
SIGSI	Función que entrega el siguiente caracter (nsbuf) sin efectuar lectura alguna.
SIGNB	Función que lee caracteres hasta poder entregar el primero distinto de blanco.
VEELNB	Función que entrega el siguiente símbolo a entregar (nsbuf), que no sea blanco.
PONCHS(tipo, pos)	Se encarga de leer un átomo (el tipo indica que clase de átomo: 0-Número, 1-Nombre, 2-Cadena), de almacenarlo como cadena en el arreglo CHARS y de entregar su posición en VARDIC (pos).
GUARDA(Token)	Almacena al token en el texto limpio.
BUSC(nombre, pos)	Revisa que el átomo (identificador o número), representado por nombre (apuntador a CHARS) exista o tenga que darlo de alta en el diccionario. En pos entrega la dirección a VARDIC que representa al átomo.
PARSER(PSP)	Función lógica que lleva a cabo el reconocimiento sintáctico. Si el análisis resultó correcto entrega el valor de verdadero. PSP es un apuntador al no-terminal de la tabla gramatical que se desea reconocer. Cabe recordar que esta rutina para trabajar se fundamenta en llamados recursivos.
PRTAR	Si el análisis sintáctico resultó correcto, el árbol sintáctico engrendrado se graba en disco, y si la opción de compilación correspondiente está activada (LARBOL es verdadero), se imprime el árbol en el listado.

PROGRAMA PRINCIPAL Se encarga de hacer el reconocimiento lexicográfico, de llamar al reconocimiento sintáctico cuando sea necesario y de finalizar la primera parte de la compilación llamando a la rutina FINLEXICO.

El archivo en que se encuentra el analizador lexicográfico-sintáctico se llama LEXICO.TEXT.

VI) ANALIZADOR SEMANTICO Y GENERACION DE CODIGO

A) INTRODUCCION

La segunda fase o programa del compilador consiste precisamente del analizador semántico, el cual se encarga de verificar que se cumplan las reglas semánticas del lenguaje (como es la utilización correcta de los identificadores según el contexto donde aparezcan; que no haya declaraciones después de la primera proposición; que no haya proposiciones compuestas sin cerrar, etc) y también le corresponde la tarea de generar código.

En este capítulo se describirán los aspectos más importantes que comprenden a la semántica y simultáneamente se irá exponiendo la estructura y elemento que conforman al código, que, como ya antes se ha mencionado, constituye un programa escrito en PASCAL, el cual hace uso de una serie de rutinas de biblioteca.

B) ESTRUCTURA Y FUNCIONAMIENTO DEL ANALIZADOR SEMANTICO

La principal característica del analizador semántico, como ya se ha mencionado es que se guía por medio de llamados a rutinas, que vienen contenidos en los árboles sintácticos.

La estructura del programa consiste en un conjunto de rutinas, las que son llamadas primeramente desde el programa principal y en forma secundaria, entre ellas mismas.

El programa principal se encarga de leer cada árbol sintáctico y al ir encontrando nodos especiales que denotan a alguna de las rutinas semánticas, esta es ejecutada y se continúa rastreando el árbol hasta alcanzar el final del mismo.

El árbol sintáctico consiste de tres arreglos lineales, en los cuales hay uno, el llamado arreglo de alternativas (A), que contiene números enteros. Cada uno de estos números, de acuerdo al rango al que pertenezca, tiene determinado significado.

Si el valor es negativo, se tiene una opción de compilación referente al análisis semántico. El resto de los nodos especifican la opción y los parámetros de la misma.

Si el valor es positivo pero menor a 4000, el nodo representa una de dos cosas: el número de alternativa o producción que se determinó para el símbolo no-terminal representado por dicho nodo, o bien, es un apuntador al diccionario de átomos y representa un identificador, un número o una cadena. La interpretación del valor corre a cargo de la rutina semántica respectiva y debe coincidir con el papel del símbolo no-terminal en la gramática.

Finalmente, si el valor no está en los rangos anteriores, es decir, es mayor o igual a 4000, entonces el nodo representa a una rutina, la que es ejecutada desde el programa principal. Así pues, el contenido del nodo mismo es un pseudo apuntador a la rutina semántica que debe analizar al árbol sintáctico o parte del mismo.

Veamos un ejemplo de como se utilizan los llamados semánticos.

Para la línea en el programa fuente:

REAL X, YY, I, FINARCHIVO; se generará el siguiente árbol sintáctico:

1	[illegible]	3	4	5	6	7	8	9
PRINC	[illegible]	F	DECL	[illegible]	L DE	1020	RD5	1020
[illegible]	2	1	2	4011	1	47	1	48
[illegible]	3	15	15	6	15	8	15	10

10	11	12	13	14
[illegible]	1020	RD5	1020	RD5
1	[illegible]	1	[illegible]	2
[illegible]	[illegible]	15	14	15

El árbol consiste de: números de nodo (linea I), nombres de símbolos no-terminales representados (linea II), números de alternativa, apuntadores al diccionario de átomos o llamados semánticos (linea III) y apuntadores a otros nodos (linea IV).

El analizador sintáctico encuentra en la gramática a los llamados semánticos conforme va procesando al texto limpio y en ese momento inserta un nodo con dicho llamado en el árbol sintático*.

Para el ejemplo presentado, la gramática contiene para la declaración de reales:

```
<DECL> :: = ... |
              "REAL" <4011> <L DE ID> |...
<L DE ID> :: = <IDENTIF> <RD5>
<RD5> :: = "," <IDENTIF> <RD5> | e
```

El analizador semántico recorre el árbol buscando los llamados semánticos y ejecutando el código correspondiente. Veamos un segmento del analizador.

```
WHILE AP < = APSUP DO BEGIN
(* AP APUNTA AL ARBOL
   APSUP ES EL ULTIMO NODO EN EL ARBOL *)
   OP : = A[AP];
   IF OP > = 4000 THEN BEGIN
   CASE OP OF
```

* En la sección C.2 del Capítulo V de esta tesis se explica con mayor detalle el proceso de construcción del árbol sintáctico.

```
            4010:  (*DECLARA CONSTANTES *)
            4011:  (*DECLARA ESCALARES *)
            :
            END;
            END ELSE AP: = AP + 1;
        END; (* WHILE *)
```

La rutina DECEST se encarga de examinar la lista de identificadores y de declararlos.

De esta forma se evita tener que revisar todo el árbol, nodo a nodo.

Además la estructura de la semántica no depende de la gramática y cambios en ésta, acarrean solamente cambios en algunas rutinas semánticas y posiblemente, la incorporación de nuevo código.

En el Apéndice D, se pueden apreciar los llamados semánticos que se encuentran implantados hasta el momento, el código ejecutado por cada uno de ellos y el papel que cumplen dentro del analizador semántico.

Pasemos a ver ahora el diagrama de funcionamiento de la semántica.

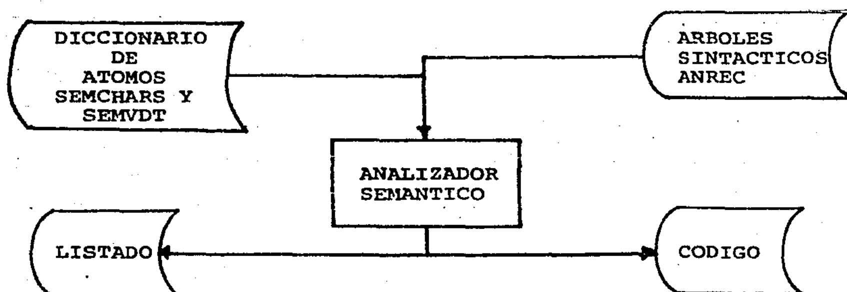


Figura 6.1 Diagrama del Analizador Semántico.

El analizador semántico primeramente carga al diccionario de átomos, constituido por los arreglos SEMCHARS y SEMVDT, que formarán parte de la tabla de símbolos. El diccionario contiene a los identificadores, números y cadenas aparecidas en el programa fuente.

A continuación el analizador se dedica a leer cada árbol sintáctico y a procesarlo. A la vez construye la tabla de símbolos, da forma a otras estructuras de datos, valida que la frase cumpla las reglas semánticas del lenguaje y emite el código correspondiente. Además se emite un listado que contiene los errores encontrados (cada error indica en que número de linea apareció y así, teniendo el listado del analizador lexicográfico se puede determinar el lugar exacto), si está activada la opción de monitoreo, se arroja una reseña detallada del análisis semántico. Si el monitoreo y el código se dirigen al mismo dispositivo (pantalla o impresora), el código aparece entre la reseña del análisis.

El análisis finaliza al encontrar el "FIN" del programa (la proposición FIN;) o al no poder leer más árboles.

Como el código es un programa en PASCAL, consistente principalmente en llamados a rutinas de biblioteca, entonces debe compilarse y ligarse a las rutinas de biblioteca que viven en la biblioteca del sistema operativo de PASCAL, (SYSTEM. LIBRARY) y entonces sí, ya puede ejecutarse.

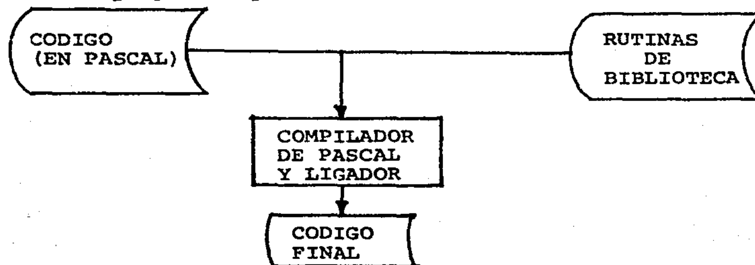


Figura 6.2. Procesamiento del Código en Pascal.

Como las cadenas de caracteres usadas en el programa fuente no se escriben directamente al código como ocurre en los números reales y enteros, al momento de ejecutar al código final se debe cargar el archivo SEMCHARS, que es el que contiene, entre otras cosas, dichas cadenas de caracteres contenidas en el programa fuente.

Si al correrse el código final, no se puede leer el archivo SEMCHARS, entonces la ejecución se llevará a cabo, tal y como si todas las constantes y variables de tipo texto dispuestas en el programa fuente, tuvieran asignadas cadenas nulas.

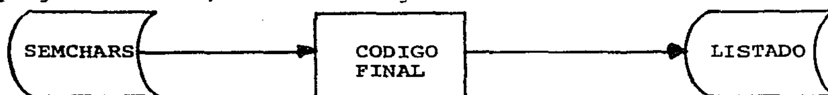


Figura 6.3. Ejecución del Código Final.

Dado que todavía no se han implantado proposiciones de entrada y salida, el código final emite un listado, por el cual se muestra una narración detallada de todas las operaciones y movimientos que se van realizando, así como de los errores de ejecución que vayan ocurriendo.

Como el código generado es un programa en PASCAL, éste puede modificarse para darle una entrada al programa o para darle una salida diferente.

Pasemos enseguida a revisar la tabla de símbolos.

C) LA TABLA DE SIMBOLOS

1) Organización de la Tabla de Símbolos.

La Tabla de Símbolos consiste de un conjunto de estructuras que viven completamente en la memoria principal. La parte que se encarga de almacenar los descriptores es una tabla (llamada DES) que se administra en forma de stack siguiendo la característica de estructura de bloques que tiene el lenguaje. De esta manera, las variables globales viven en el fondo del stack y las variables del bloque que esté procesando la semántica en un momento

dado, viven en o cerca del tope del stack.

Cuando se ingresa a un nuevo bloque, la posición del stack de descriptores en donde estará la primera variable de dicho nivel es marcada. Esto se hace con el fín de poder diferenciar los descriptores de distintos bloques. Esas marcas son realmente apuntadores que viven en un arreglo llamado NIVELS, en donde para cada nivel o bloque se indica en donde comienzan a residir los descriptores correspondientes a dicho bloque.

Veamos la siguiente figura.

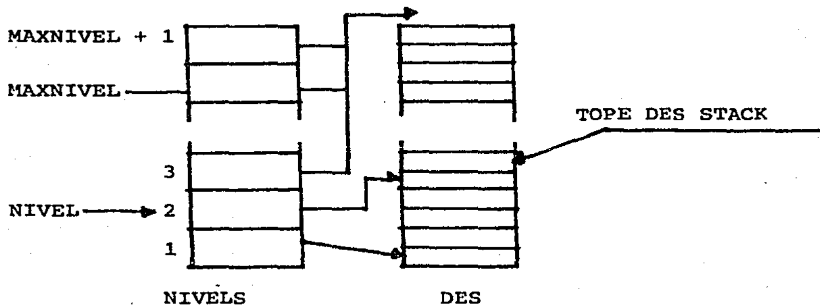


Figura 6.4. Stack de Descriptores.

NIVELS[1] apunta a donde comienzan los descriptores de las variables globales. NIVELS[2] apunta a donde comienzan los de la rutina que en ese momento se está procesando. Los demás niveles apuntan afuera del stack. El número de nivel que se está analizando vive en la variable escalar NIVEL. NIVELS contiene el apuntador MAXNIVEL + 1, correspondiente al máximo nivel permitido más uno, para que con sólo comparar la posición de un descriptor con los valores de NIVELS se pueda determinar fácilmente a que bloque corresponde dicho descriptor. Es decir, el nivel más bajo que apunte a una posición mayor a donde vive el descriptor va a ser el bloque inmediato superior a quién pertenece la variable representada.

Al encontrarse el compilador una declaración, se determinan los atributos de la variable y se almacenan en el tope del stack de descriptores, pero antes se verifica que la variable aludida no se haya declarado ya, en el mismo bloque. Para tener un fácil y rápido acceso a los datos de las variables asociadas a un mismo identificador, se diseñaron una serie de estructuras que a continuación se describen.

Todos los descriptores que corresponden a variables que comparten el mismo identificador están asociados a una lista constituida por dos arreglos. Un arreglo (LI) contiene las ligas hacia los siguientes nodos y otro arreglo (INFO), contiene los apuntadores hacia el stack de descriptores. Esta lista también es administrada en forma de stack y en el fondo vive la variable más vieja.

Las cabezas de las listas viven en el arreglo llamado LIG, el cual contiene apuntadores hacia los primeros nodos de cada lista. (Ver la figura 6.5 más abajo).

Para tener acceso a la información de alguna variable, tomamos al nodo del árbol sintáctico que representa al identificador de la variable buscada y leemos el valor que tiene en el arreglo de alternativas. Ese valor es un apuntador precisamente al arreglo LIG.

Falta por describir la parte de la Tabla de símbolos que contiene las cadenas que constituyen a los identificadores de todas las variables en el programa fuente. Esta estructura consiste de dos arreglos, un arreglo de caracteres (CHARS) que contiene a las cadenas junto con sus respectivas longitudes y un arreglo de enteros (VARDIC) que contiene apuntadores hacia cada cadena en CHARS y además, es un arreglo paralelo a LIG. De esta manera, a partir del árbol sintáctico puedo conocer el identificador y los atributos de una variable dada. Esos dos arreglos son llenados de información al iniciar el análisis semántico, a partir de los archivos dejados por el analizador lexico-sintáctico: SEMCHARS y SEMVDT.

Anteriormente se mencionó que la lista de descriptores se maneja en forma de stack, veamos a continuación, como se administra.

Cada vez que hay una declaración se verifica que el identificador involucrado no se haya declarado ya antes en el mismo bloque, para ello basta con fijarse si el valor del primer nodo de la lista correspondiente a dicho identificador es mayor o igual al valor que tiene el arreglo NIVELS en el bloque en revisión, en cuyo caso se emite el error correspondiente. Claro está, que si la lista está vacía, entonces no se ha declarado dicha variable.

En caso de no haber ocurrido error se crea un nodo que apuntará al tope del stack de descriptores, a donde quedarán los atributos de la variable. El nuevo nodo se situará a la cabeza de la lista.

Al finalizar un bloque o nivel, se eliminan todos los descriptores correspondientes a dicho bloque. Cada descriptor apunta a los arreglos LIG y VARDIC, de tal manera que resulta sencillo eliminar el primer nodo de cada una de las listas involucradas al actualizar la tabla. También se debe actualizar el contenido del arreglo NIVELS.

Veamos un ejemplo:

Supongamos que tenemos el programa:

```
              INICIO
                REAL X;
                RUTINA R
                 INICIO
                   TEXTO X, CAD;
        (*)
                 FIN;
        (**)
              FIN;
```

en los árboles tendré:

A (arreglo de Alternativas)

REAL X; | A |

RUTINA R
 INICIO; | B |

TEXTO X, CAD | A | C |

A,B,C son los apuntadores a las cabezas de listas (LIG) y al diccionario de identificadores (VARDIC).

La Tabla de Símbolos al momento (*) estará como sigue:

Figura 6.5 Tabla de Símbolos.

Al salir de la Rutina R, en el momento (**), la Tabla de Símbolos quedará:

Figura 6.6 Tabla de Símbolos.

† La segunda columna de la tabla DES representa los apuntadores al arreglo VARDIC (a las localidades A, B y C de VARDIC).

2) Descriptores

La representación de una variable en la Tabla de Símbolos se compone de dos partes: el descriptor primario y un grupo de descriptores adicionales, que puede ser vacío. Los descriptores adicionales se utilizan cuando la variable es compuesta y representan los parámetros de una rutina, los campos de una estructura (gráfica, tabla, histograma) o las dimensiones de un arreglo.

Cada descriptor primario se compone de cinco campos:

i) Apuntador a la tabla de cabezas de listas (LIG) y al diccionario de identificadores (VARDIC).

ii) Tipo de la variable:

1 - Texto
2 - Real
3 - Gráfica
5 - Tabla
7 - Histograma
10 - Archivo
11 - Formato
12 - Rutina
13 - Función

iii) Forma de la variable:

3 - Constante
4 - Variable escalar o parámetro por valor
5 - Escalar como parámetro por referencia
7 - Gráfica
8 - Gráfica como parámetro por referencia
10 - Cuerpo de Gráfica
16 - Arreglo Renglón
17 - Arreglo Columna, con más de una dimensión o arreglo de textos
18 - Arreglo Renglón como parámetro por referencia
19 - Arreglo Columna, con más de una dimensión o arreglo de textos como parámetro por referencia
25 - Tabla
26 - Tabla como parámetro por referencia
13 - Cuerpo de Tabla
28 - Histograma
29 - Histograma como parámetro por referencia
22 - Cuerpo de Histograma
31 - Rutina

iv) Localidad

Indica la localidad o posición relativa al inicio del bloque o nivel, en donde estará o comenzará a almacenarse la

variable o estructura de datos.

v) Número de descriptores adicionales. En el caso de variables de tipo escalar o rutinas sin parámetros o archivos o formatos, este campo vale cero.

En caso de que existan descriptores adicionales, el significado de la información contenida, depende de cada variable o estructura de datos, a excepción del campo (1) que en los descriptores primarios apunta a los arreglos LIG y VARDIC y en los descriptores adicionales siempre es cero.

A continuación se muestran los descriptores de cada una de las estructuras de datos que pueden declararse en el lenguaje.
Omito el campo (1), pues como ya expliqué, sólo se emplea en los descriptores primarios y siempre con igual finalidad.

Constante:

2) Real (1) o Texto (2).
3) Constante (3).
4) Localidad en memoria, si es Texto o Apuntador al arreglo de caracteres (CHARS); a donde está escrito el número, si es Real.
5) Cero.

Escalar:

2) Real (1) o Texto (2).
3) Variable simpe o parámetro por valor (4), por referencia (5).
4) Localidad en memoria. Si es un parámetro pasado por referencia, localidad donde estará la dirección de la variable.
5) Cero.

Arreglo:

Primera Parte.

2) Real (1) o Texto (2).
3) Arreglo Renglón (16). Columna, con más de una dimensión o arreglo de textos (17). Si es parámetro por referencia, (18) ó (19) respectivamente.

4) Localidad en memoria. Si es parámetro por referencia, localidad que tiene la dirección del arreglo.
5) Número de dimensiones. Si es parámetro por referencia contiene un uno.

Segunda Parte.

Por cada dimensión:

2) Valor del límite inferior*.
3) Valor del límite superior.
4 y 5) Cero.
Si es un parámetro por referencia, el campo 2 contiene el número de dimensiones y el resto del descriptor contiene ceros.

En la sección D del presente capítulo se describen los descriptores dinámicos de los arreglos, que complementan la información de la Tabla de Símbolos y son un elemento importante de la implantación y diseño del compilador.

Gráfica:

Primera Parte.

2) Gráfica (3).
3) Gráfica (7). Si es parámetro por referencia (8).
4) Localidad en memoria donde empieza el primer campo (TITULO). Si es parámetro por referencia, localidad donde está la dirección de la estructura.
5) Siete. Si es parámetro por referencia, cero.

Segunda Parte.

Para los campos TITULO, TITREN y TITCOL respectivamente:
2) Texto (2)
3) Variable simple (4)
4) Localidad en memoria.
5) Cero.

---

* Por razones de implantación que más tarde se expondrán, el límite inferior siempre deberá ser uno.

Para el campo CUERPO:

2) Real (1).
3) Cuerpo de Gráfica (10).
4) Localidad en memoria.
5) Cero.

Dimensiones:

2) Valor del primer número o constante en la declaración.
3) Valor del segundo número o constante en la declaración.
4 y 5) Cero.

Dos descriptores para los Máximos y Mínimos de las abscisas y ordenadas, respectivamente.

2) Localidad del Máximo.
3) Localidad del Mínimo.
4 y 5) Cero.

Tabla:

Primera Parte:

2) Tabla (5).
3) Tabla (25) o parámetro por referencia (26).
4) Localidad en memoria donde empieza a almacenarse el primer campo: TITULO. Si es parámetro por referencia, localidad que contiene la dirección de la estructura.
5) Nueve. Si es parámetro por referencia es cero.

Segunda Parte:

Para los campos TITULO, TITREN y TITCOL es igual que en la estructura Gráfica.

Campo Cuerpo:

2) Real (1).
3) Cuerpo Tabla (13)
4) Localidad en memoria.
5) Cero.

Para las dimensiones, igual que en la estructura Gráfica.

Campos NOMREN y NOMCOL:

2) Texto (2).
3) Arreglo de textos, columna o con más de una dimensión (17).
4) Localidad en memoria.
5) Cero.

Campos MARCOREN y MARCOCOL:

2) Real (1).
3) Arreglo columna, con más de una dimensión o texto (17).
4) Localidad en memoria.
5) Cero.

Histograma:

Primera Parte:

2) Histograma (7).
3) Histograma (28) o parámetro por referencia (29).
4) Localidad en memoria del primer campo que es el TITULO. Si es parámetro por referencia, localidad donde está la dirección de la estructura.
5) Cuatro. Si es parámetro por referencia, cero.

Segunda Parte:

Campo TITULO, igual que en las otras dos estructuras.

Campo NOMREN, igual que en el caso de la estructura TABLA.

Campo CUERPO:

2) Real (1).
3) Cuerpo Histograma (22).
4) Localidad en memoria.
5) Cero.

Dimensión del Histograma.

2) Número de frecuencias.
3,4 y5) Cero.

Rutina:
Primera Parte:

2) Con tipo (12). Sin tipo (13).
3) Rutina (31).
4) Nombre interno.
5) Número de parámetros (si es con tipo, se agrega uno más - al principio - con el tipo de la rutina, que es Real. En la cuarta posición de ese descriptor se coloca la localidad donde se dejará el valor de la función).

Segunda Parte:

Por cada parámetro:

1) Cero.
2) Tipo.
3) Forma.
4) Localidad en memoria de la dirección de la estructura de datos para parámetros pasados por referencia. Si es un escalar real pasado por valor, localidad en donde vive.
5) Número de dimensiones, en el caso de arreglos.

Archivo:

2) Archivo (10).
3) Cero.
4) Nombre interno.
5) Cero.

Formato:

2) Formato (11).
3) Número de textos que lo componen.
4) Apuntador al arreglo de caracteres.
5) Cero.

En el presente trabajo no se alcanzan a manejar dentro del compilador, los descriptores de rutinas, parámetros de rutinas, archivos y formatos.

Cuando el monitor del análisis semántico está prendido, se muestran los apuntadores al arreglo INFO y al stack de descriptores, así como el contenido de los descriptores. Enseguida muestro un ejemplo:

```
%  EJEMPLO DE DECLARACIONES ;

INICIO
$$D+
   CONSTANTE S = 'CONSTANTE',
             N = 12;
   REAL R1, R2 ;
   ARREGLO TEXTO ARR[1:10, 1:20 ] ;
   TABLA TAB[ 10, 5 ];
$$D-
FIN;
```

Figura 6.7 Programa que ejemplifica declaraciones.

El analizador semántico al procesar las líneas anteriores emite el siguiente listado:

```
PONNIV :  APVDT 45 APDIS 1 INFO[APDIS] 1 ____________ S
DESCRI  : APVDT 45 APDES 0 CP 2 CS 3 CT 1 CC 0
PONNIV :  APVDT 46 APDIS 2 INFO[APDIS] 6 ____________ N
DESCRI  : APVDT 46 APDES 5 CP 1 CS 3 CT 275 CC 0
PONNIV :  APVDT 48 APDIS 3 INFO[APDIS] 11 ___________ R1
DESCRI  : APVDT 48 APDES 10 CP 1 CS 4 CT 17 CC 0
PONNIV :  APVDT 49 APDIS 4 INFO[APDIS] 16 ___________ R2
DESCRI  : APVDT 49 APDES 15 CP 1 CS 4 CT 18 CC 0
PONNIV :  APVDT 50 APDIS 5 INFO[APDIS] 21 ___________ ARR
DESCRI  : APVDT 50 APDES 20 CP 2 CS 17 CT 19 CC 2
DESCRI  : APVDT 0 APDES 25 CP 1 CS 10 CT 0 CC 0
DESCRI  : APVDT 0 APDES 30 CP 1 CS 20 CT 0 CC 0
PONNIV :  APVDT 54 APDIS 6 INFO[APDIS] 36 ___________ TAB
DESCRI  : APVDT 54 APDES 35 CP 5 CS 25 CT 3222 CC 9
DESCRI  : APVDT 0 APDES 40 CP 2 CS 4 CT 3222 CC 0
DESCRI  : APVDT 0 APDES 45 CP 2 CS 4 CT 3238 CC 0
DESCRI  : APVDT 0 APDES 50 CP 2 CS 4 CT 3254 CC 0
DESCRI  : APVDT 0 APDES 55 CP 1 CS 13 CT 3270 CC 0
DESCRI  : APVDT 0 APDES 60 CP 10 CS 5 CT 0 CC 0
DESCRI  : APVDT 0 APDES 65 CP 2 CS 17 CT 3323 CC 0
DESCRI  : APVDT 0 APDES 70 CP 2 CS 17 CT 3485 CC 0
DESCRI  : APVDT 0 APDES 75 CP 1 CS 17 CT 3567 CC 0
DESCRI  : APVDT 0 APDES 80 CP 1 CS 17 CT 3579 CC 0
MINIMA MEMORIA DISPONIBLE : 3993
**** ACABO LA COMPILACION ***  0 ERRORES **
```

Figura 6.8 Listado que emite el Analizador Semántico, correspondiente al programa de la figura 6.7

En los renglones PONNIV se muestra:

APVDT : Apuntador a VARDIC y LIG.

APDIS : Apuntador a INFO.

INFO[APDIS] : Apuntador al stack de descriptores (DES).

En DESCRI se muestra:

APDES : Tope del stack de descriptores. Aumenta de cinco en cinco porque el stack sólo consiste de enteros y se requieren cinco enteros para representar cada descriptor*.

APVDT : El primer campo del descriptor. Apuntador a LIG y VARDIC en los descriptores primarios y cero en los descriptores adicionales.

CP,CS,CT,CC: Los campos restantes del descriptor que realmente contienen los atributos de las variables.

D) ADMINISTRACION DE MEMORIA

En esta parte se describe como está concebida la memoria para el código, como se representan las distintas estructuras o variables que pueden declararse en el lenguaje estadístico y como se generan e inicializan dichas representaciones.

1) Stack de Memoria

La memoria del programa objeto reside en un arreglo lineal llamado MEM. Cada celda de MEM puede ser vista como un real o como cuatro caracteres, de tal manera que el mismo espacio puede emplearse para representar a los dos tipos elementales de datos que hay en el lenguaje: caracteres y números reales.

La declaración del arreglo de memoria consiste de lo siguiente:

```
VAR MEM : ARRAY [MINMEM..TAMMEM] of TIPMEM;
```

donde MINMEM y TAMMEM son constantes y el tipo de cada celda es como sigue:

```
TYPE
    TMULT = (RE, CARD);
```

---

* Este diseño del stack de descriptores se debe a que se mantuvo la forma que se tenía en la versión original, en FORTRAN.

```
TIPMEM = PACKED RECORD
           CASE TMULT OF
             RE: (R: REAL);
             CAR: (C: PACKED ARRAY [0..3]
                      OF CHAR)
           END;
```

Al empacar al registro o "record" se elimina al campo descriminante que indica que tipo es el empleado en cada celda, de tal forma que cada celda ocupa exactamente 32 bits, ya sea un real o cuatro caracteres. Como puede verse no hay desperdicio de bits.

El acceso a la memoria es sencillo. MEM[I].R para tomar al real en la celda I. MEM[I].C[J] para el caracter J de la celda I.

Entre los requerimientos que ha presentado el compilador no se ha necesitado tener acceso a la memoria a nivel de bits, por lo que no resulta preponderante determinar el orden en que el compilador de PASCAL distribuye a los cuatro caracteres dentro del registro. Lo único que se buscó fué que cada celda de la memoria se pueda tomar en cualquier momento como un real o como cuatro caracteres independientes entre sí y que no exista el campo descriminante o selector que indique el tipo utilizado en el registro.

Debido a que el lenguaje estadístico vive en un ambiente de estructura de bloques, la memoria se administra en forma de stack. Para complementar al stack de memoria, se dispone de un arreglo llamado DISPLAY, el cual funciona como un grupo de registros de despliegue "display". Cada elemento de DISPLAY apunta a la activación actual del bloque o nivel lexicográfico en el stack de memoria, a que corresponda. (Ver la figura 6.9).

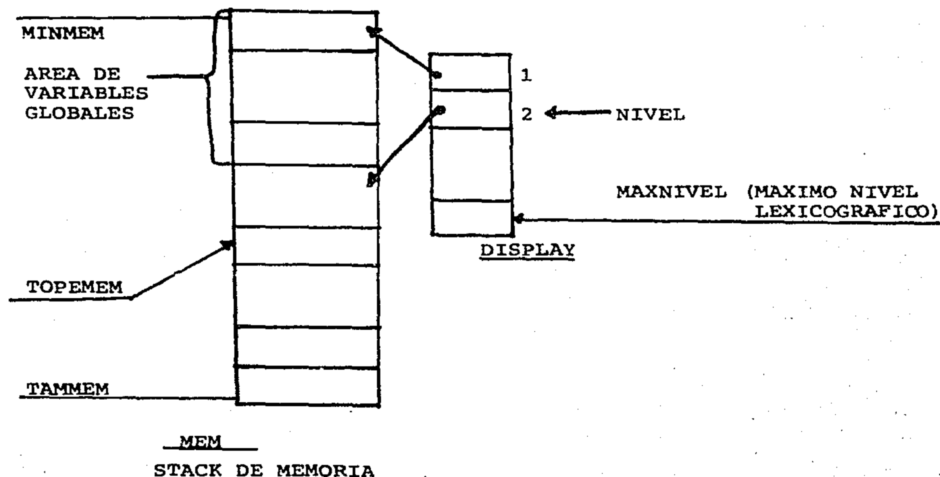


Figura 6.9 Stack de Memoria.

De esta manera la dirección en memoria de cada variable se conforma a partir de dos datos: el valor base de cada bloque o nivel lexicográfico que está contenido en el arreglo DISPLAY y un desplazamiento relativo a dicho valor base del bloque y que viene contenido en el descriptor de la variable. Es decir, el desplazamiento se determina en compilación y se graba en el código cuando se encuentren referencias a dicha variable.

Como ya lo mencioné en la sección dedicada a la Tabla de Símbolos, la variable NIVEL es quien especifica el bloque que se esté ejecutando en cada momento.

El tope del stack de memoria se llama TOPEMEM.

No existe un apuntador al código, puesto que el programa objeto es un programa en PASCAL que intrínsecamente lleva el control de la ejecución.

Por último, el stack es inicializado, por el momento, sólo al comenzar la ejecución del programa objeto, debido a que no se ha

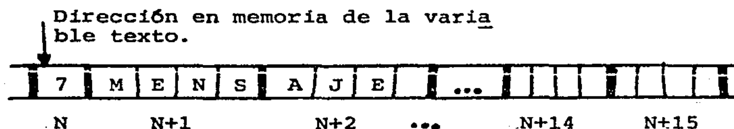
puesto énfasis en un procedimiento óptimo y eficiente de inicialización de memoria, pues se considera que mejor convendría hacerlo hasta tener implantado el manejo de rutinas o dejarlo como mejora al compilador.

El stack se inicializa en ceros, así las cadenas comienzan como nulas y los números en ceros.

2) Representación de Variables

Un escalar real se representa mediante una sola celda de memoria. Su localidad se determina como ya lo mencioné, con el desplazamiento en su descriptor y con el valor base en memoria del bloque a que pertenezca.

Un escalar texto ocupa dieciseis celdas de memoria. Quince celdas se destinan para almacenar caracteres, lo que permite guardar hasta sesenta caracteres, que es la longitud máxima que puede tener una cadena. La celda restante, que es la primera, se emplea para indicar la longitud de la cadena y entonces se maneja como número real. (Ver la figura 6.10).



MEM[N].R = longitud de la cadena

MEM[N+1]. C[1] = primer caracter de la cadena

MEM[N+15]. C[3] = último de los sesenta caracteres para almacenar una cadena.

N se calcula de la siguiente forma:

N = DISPLAY [NIVEL] + DESPL, donde DESPL es la localidad que contiene el descriptor y NIVEL es el bloque a que pertenece la variable.

Figura 6.10 Representación de una Variable Texto.

Cuando la longitud es cero la cadena se considera como nula.

Como el lenguaje está orientado al proceso numérico de datos y las cadenas se utilizan más que nada para escribir reportes, con el fín de facilitar el manejo de ellas se resolvió dar una longitud máxima y un espacio fijo para almacenar a las cadenas.

Cabe mencionar que en la versión en FORTRAN se había decidido almacenar sólo dos caracteres por celda, debido a restricciones del lenguaje anfitrión. Se requerían por tanto, del doble de celdas que en esta versión. (Cada celda en ambas versiones contiene exactamente treinta y dos bits).

Las constantes de tipo real carecen de representación en memoria. Cada vez que se invoca a una constante real durante la compilación, se toma primeramente su descriptor, el cual contiene la posición en el diccionario de identificadores, cadenas y números en donde está definida, es decir, donde está la cadena de caracteres numéricos que definen a dicha constante. A continuación se llama a una rutina llamada VALOR que entrega finalmente el valor en decimal de la constante. Este valor se graba directamente al código, es decir, se escribe una auténtica literal en el código objeto PASCAL.

Las constantes tipo texto, al contrario de las anteriores, si requieren vivir en la memoria del programa objeto, debido a que es costoso en código estar generando instrucciones que asignen una cadena a una área del stack de memoria o a otra estructura de datos, cada vez que se solicite una constante texto. Para evadir esa anomalía de generar demasiado código, se ideó un procedimiento más económico. A cada constante se le asigna un espacio en el stack de memoria tal y como si fuera una variable escalar tipo texto (la localidad en memoria se guarda en el descriptor de la constante) y se genera una rutina (LECCAD) que se encargará de leer la cadena del diccionario y dejarla en memoria, en la dirección mencionada. Esta rutina se ejecutará antes que las líneas correspondientes a las proposiciones del bloque respectivo. El diccionario que menciono forma parte del mismo diccionario de átomos que vive en compilación. Para más claridad, es la tabla que contiene las cadenas que conforman los átomos del programa fuente (SEMCHARS) y que como ya mencio

né antes requiere ser leída por el programa objeto.

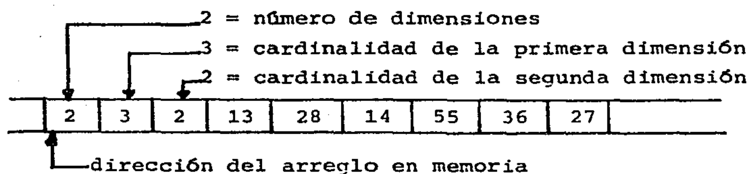
Por supuesto, que si el archivo SEMCHARS no está presente al ejecutarse el código, las constantes actuarán como cadenas nulas.

Conviene comentar como se manejan las cadenas que aparecen en el programa fuente. Como se supone que se tiene acceso al archivo SEMCHARS durante la ejecución del código, cada vez que aparece una cadena en una proposición se genera un llamado a la rutina LEC2CAD que leerá el texto y lo dejará en el tope del stack de memoria, para que sea tomado por otro procedimiento, con catenación o asignación a una variable.

Pasemos ahora a ver como se representan los arreglos.

Cualquier arreglo, independientemente de su tipo, contiene al inicio de su representación una descripción del mismo. Es decir, los valores de los arreglos quedan precedidos de un descriptor dinámico. Cada elemento de la descripción ocupa una celda de memoria. El primer elemento indica el número de dimensiones del arreglo. Cuando se tiene una sola dimensión se pone un 1 para un arreglo columna o tipo texto y un -1 para un arreglo renglón. Si N es el número de dimensiones, las siguientes |N| celdas contiene las magnitudes de cada dimensión. (Ver las figuras 6.11 y 6.12).

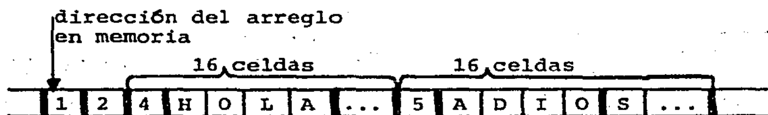
Enseguida vienen las celdas destinadas a los elementos del arreglo. Una celda por elemento para arreglos reales y 16 celdas por elemento para arreglos texto. Se disponen de tantas celdas, como se necesiten para contener el número exacto de elementos del arreglo. Los elementos se almacenan por renglones, es decir, los elementos de la dimensión más a la derecha son los que están contiguos.



Este arreglo es bidimensional, con 3 renglones y 2 columnas y representa a la siguiente matriz:

$$\begin{pmatrix} 13 & 28 \\ 14 & 55 \\ 36 & 27 \end{pmatrix}$$

Figura 6.11 Representación de un Arreglo Real.



Este arreglo tipo texto consta de una dimensión, con dos elementos y ocupa en total 34 celdas. Las cadenas que contiene son HOLA y ADIOS.

Figura 6.12 Representación de un Arreglo Texto.

La descripción que contienen los arreglos en ejecución es generada en compilación cuando se procesan las declaraciones de arreglos. Para ello se disponen de dos rutinas: DEC1ARR y DEC2ARR. La primera rutina, almacena a partir de la localidad en memoria asignada al arreglo en compilación, el número de dimensiones y la cardinalidad de la primera dimensión. La segunda rutina almacena la cardinalidad de una siguiente dimensión y se llama tantas veces como subsecuentes dimensiones hayan.

Hay varias modificaciones que se hicieron al proyecto original para el manejo de arreglos. Primeramente, el límite inferior en los arreglos siempre debe ser 1. En la concepción original, la declaración de un arreglo puede admitir cualquier valor para el límite inferior de las dimensiones, pero por otra parte, si

el resultado de una expresión es un arreglo, se considera que el límite inferior en cada dimensión es 1. Se consideró que esta pauta podía acarrear más confusiones que comodidad al usuario y además dificultaría la implantación. Por ejemplo, consideremos a los arreglos unidimensionales A[-200;-1] y B[-99:100] y analicemos la proposición B=A+B. Al hacer la adición, como tengo arreglos producto de declaraciones, los elementos de los sumandos se numeran según los límites en la declaración. Entonces A+B es un arreglo de 301 elementos que van de 1 a 301 y la pregunta sería: ¿Qué pasa en la asignación?. Ya sea que ésta se haga respetando los índices de B+A y B o sea haga elemento a elemento, se obtendría un resultado muy independiente de los datos que originalmente se tenían.

Como además en estadística, los datos casi siempre se numeran desde 1, se tomó la convención de que los arreglos siempre se declaren con el límite inferior igual a 1 y que en los arreglos producto de expresiones, sus elementos se numeren desde 1.

Las otras modificaciones residen en que la cardinalidad mínima permitida para las dimensiones en la declaración de arreglos debe ser dos y que la operación de dimensionamiento* de arreglos se haga al momento de terminar una operación que involucre uno o dos arreglos y no al tomar un arreglo como ocurría antes. La ventaja es que al exigir un mínimo tamaño en las dimensiones, al tomar un arreglo producto de declaración no hace falta dimensionarlo y elimina posibles errores por declaraciones anómalas. Además el dimensionar al finalizar una operación permite una evaluación más segura y limpia de las expresiones.

En cuanto a las estructuras Gráfica, Tabla e Histograma, los campos que las constituyen corresponden en su mayoría a las estructuras ya descritas, a excepción del Cuerpo de Gráfica, que es una estructura de datos especial.

---

* El proceso de dimensionamiento se aplica a los arreglos y consiste en dejar a la variable con el número exacto de dimensiones que contiene, considerando que el mínimo tamaño para una dimensión es dos. Si un arreglo llega a tener sólo un elemento en todas las dimensiones, lo que realmente se tiene es una variable escalar. Para conocer con más detalle a esta operación acudir a García [G1].

Los campos TITULO, TITCOL y TITREN se manejan como variables tipo texto. Los campos NOMREN y NOMCOL son arreglos unidimensionales de tipo texto. Los campos MARCOCOL y MARCOREN son arreglos reales de una dimensión. Los máximos y mínimos de las abscisas y ordenadas de la Gráfica se representan como variables reales escalares. El Cuerpo de Histograma es un arreglo real unidimensional, el de una Tabla es una matriz de reales y el de una Gráfica es una estructura especial que contiene todos los puntos de la Gráfica y aún no se ha implantado. Ello se debe a que se considera que es más conveniente dejar esta labor para cuando se tenga ya implantada la parte de entrada-salida del lenguaje.

El orden en que se distribuyen los campos, es el mismo que se muestra en los descriptores de las estructuras, en una sección anterior.

Siguiendo la filosofía en el manejo de arreglos, ya muy comentada, no puede haber tablas, ni gráficas de un sólo renglón o una sola columna, ni histogramas con una sola clase.

Todos los llamados para inicializar los descriptores de los arreglos y para inicializar los espacios para las constantes tipo texto se hacen antes de ejecutar las líneas correspondientes a las proposiciones del bloque correspondiente y están dentro de una rutina llamada INI_{jk}, donde j es el nivel o bloque correspondiente y k es el número de rutina dentro de ese bloque. Así para inicializar a las variables globales se crea la rutina INI11.

La declaración de las variables, así como la emisión de código con los llamados a rutinas que lean cadenas e inicialicen descriptores dinámicos de arreglos corresponden a los siguientes llamados y rutinas semánticas:

4010.- DECCTE. Declara una constante real o texto.

4011.- DECESC. Declara una lista de escalares reales o de escalares textos.

4012.- DECARREGLOS.- Declara una lista de arreglos.

4013.- DECEST(FALSE). Declara una lista de Gráficas.

4014.- DECEST(TRUE). Declara una lista de Tablas.

4015.- DECHIST. Declara una lista de Histogramas.

A continuación se presenta el programa objeto correspondiente al programa fuente de la figura 6.7 y cuya reseña semántica se presenta en la figura 6.8, donde se muestra el contenido de los descriptores.

```
(*$S++*)
(*$G+*)
PROGRAM ENGEST;
USES TRANSCEN, CODGRAL, INICOD;
LABEL 1, 2, 3, 4, 5, 6, 7, 8, 9, 10;
PROCEDURE INI11;          ——— RUTINA DE INICIALIZACION DE
BEGIN                          VARIABLES GLOBALES (NIVEL 1)
INICIALI;
NIVEL:=1;
LEECAD(DISPLAY[1]+1,261,9);   ——— CARGA CONSTANTES
DEC1ARR(19,2,10);  }
DEC2ARR(20);       }  ——— ARREGLO ARR
DEC1ARR(3270,2,10);}
DEC2ARR(5);        }
DEC1ARR(3323,1,10);}
DEC1ARR(3485,1,5); }  ——— TABLA TAB
DEC1ARR(3567,1,10);}
DEC1ARR(3579,1,5); }
TOPEMEM:=DISPLAY[1]+3586;
END; (* INI11 *)
BEGIN
INI11;
CLOSE(LST,LOCK);
END.
```

Figura 6.13 Programa objeto correspondiente al programa fuente de la figura 6.7 .

E) COMPILACION DE EXPRESIONES

1) Introducción

Las expresiones cumplen un papel muy importante en el lenguaje

estadístico, hasta podrían considerarse el corazón del lenguaje, pues constituyen el instrumento para efectuar cualquier cálculo con los datos, de una manera sencilla, que no dificulte la aplicación de los métodos estadísticos.

Las expresiones en el lenguaje se emplean tanto para producir resultados de tipo aritmético, como para obtener resultados de índole lógico. Según el contexto en donde aparezca, la expresión se puede interpretar en el sentido puramente numérico o bien, como una expresión booleana o lógica, como ocurre en una proposición condicional.

Se considera que una expresión es verdadera si su valor es distinto de cero y falsa si su evaluación resulta igual a cero.

En este capítulo veremos cual es el procedimiento de compilación de una expresión, que estructuras de datos se utilizan, como auxiliares; en que consiste el código generado y las estructuras de datos de que este emplea.

Comenzaremos por referirnos al análisis y evaluación de agrupaciones de operaciones binarias y unarias. Luego veremos la construcción de literales u operandos constantes en forma de arreglos y finalmente, la obtención de estructuras en forma de arreglo, a partir de otros arreglos.

2) Operaciones Binarias y Unarias

Para evaluar expresiones se dispone de dos pilas o stacks, uno para almacenar operandos y resultados temporales (STOPNDO) y otro, destinado para guardar a los operadores (STOPDOR).

La generación de código se hace directamente, es decir, se evita el paso intermedio, de traducir la expresión a notación polaca. Además, a los operadores no se les asigna precedencia alguna dentro del reconocedor semántico, pues los árboles sintácticos las contienen implícitamente, ya que la gramática de las expresiones es de precedencia de operadores.

Todo el análisis de una expresión está dirigido por llamados semánticos. Veamos el segmento de la gramática referente a la definición de expresión.

```
<Exp> :: = <Exp Simple> <R3>
<R3> :: = <Op.de Relación> <5001> <Exp Simple>
          <5025> | e
<Op. de Relación> :: = "< =" | "<>" |"<" |" >="
                               | ">" | "="
<Exp Simple> :: = <Op. Unario> <5015> <Término>
                  <5026> <R4>
<Op. Unario> :: = "+" | "-" | e
<R4> :: = <Op. Suma> <5007> <Término>
          <5025> <R4> | e
<Op. Suma> :: = "+" | "-" | "o"
<Término> :: = <Factor> <R5>
<R5> :: = <Op. Mult> <5010> <Factor> <5025>
          <R5> | e;
<Op. Mult> :: = "*" | "/" | "><" | "Y"
<Factor> :: = <Operando> <R6>
<R6> :: = "**" <5014> <Operando> <5025> |
          "[illegible]" <5019> <5026> | "'" <5020> <5026>
          | e
<Operando> :: = <5027> <Primario> <Exp Selec>
                | "NO" <Operando> <5018> <5026>
<Primario> :: = <Número> <5028> | <Texto> <5029>
                | <Identificador< <5032> <ACPV>
                | "PI" <5030> | "E" <5031>
                | "!" <Exp> "!" <5021> <5026>
                | <Arr. Explícito> |
                "(" <Exp> <Graf o Real Arr>
<Graf o Real Arr>:: = ")" | "," <Exp> "," <Exp> ")"
<ACPV> :: = "." <Op. Est> | <Zona de Par. Reales>
            | e
<Exp Selec> :: = <Exp Selec de Arr> | e
```

La gramática anterior define casi por completo a una expresión, a excepción de algunos símbolos no-terminales que en su momento serán desarrollados.

En general sólo se agregaron los llamados semánticos a la ver-

sión original de la gramática, por lo que si se desea conocer a fondo el significado de cada operador y operando se sugiere acudir a García [G1].

Los llamados semánticos se pueden dividir por su función, en tres grupos. El primer grupo contiene los llamados 5028, 5029, 5030, 5031 y 5032 que se encargan de meter un operando al stack respectivo. El operando puede ser un número, una cadena o texto, las constantes predefinidas PI o E, un arreglo explícito (las mismas rutinas que se encargan de procesarlo, meten el operando correspondiente al stack) o un identificador que puede corresponder a una variable aislada, a un campo de una estructura o al llamado de una función, con o sin parámetros.

Hay un llamado especial, el 5027 que se encarga de validar que una expresión selectiva de arreglo se pueda aplicar al operando representado por el símbolo <Primario>.

En el segundo grupo, los llamados siempre están situados a la derecha de los operadores. Su papel únicamente consiste en meter al stack de operadores el elemento correspondiente. Los símbolos 5001, 5007, 5010 y 5014 conciernen a los operadores binario y los símbolos 5015, 5018, 5019, 5020 y 5021 pertenecen a los unarios.

El tercer grupo comprende a las rutinas que se encargan de generar las operaciones binarias y unarias. Su labor comienza tomando uno o dos elementos del stack de operandos y un operador del otro stack. Validan que los operandos sean compatibles y que sea factible que la operación pueda aplicarse. Como no hay mecanismo alguno en el lenguaje que transforme datos de un tipo al otro, es posible determinar si un resultado será real o texto. Cabe indicar que la única operación permitida entre cadenas es la concatenación (+).

Finalmente se genera al código, el llamado a la rutina que se encargará de efectuar la operación respectiva. Se construye un operando llamado temporal que representa el resultado de la operación y se mete al stack de operandos.

Cuando el operador tomado, corresponde a un mas-unario no se genera llamado alguno, ni se modifica el stack de operandos.

Más tarde, se analizará la composición del stack de operandos.

El llamado 5025 corresponde a las operaciones binarias y el 5026 a las unarias.

La gramática está concebida para que implícitamente especifique las precedencias de los operadores, pues conforme los operadores se van definiendo a mayor profundidad, mayor precedencia tienen. Gracias a ello no fué necesario sofisticar la administración del stack de operadores, bastó con tener una rutina para meter un elemento al stack y otra, para sacar del stack y mandarlos al código, tal y como ya se expuso.

Los operadores de suma y multiplicación están definidos como asociativos por la izquierda, es decir la expresión a*b*c equivale a (a*b)*c.

Por otra parte, la asociatividad en la operación de potencia debe indicarse explícitamente. Por ejemplo, la expresión a**b**c producirá un error de sintaxis, debe escribirse a**(b**c) o (a**b)**c.

Si hay dos o más operaciones de comparación en una misma expresión, todas ellas deben encerrarse entre paréntesis. Por ejemplo, a<b Y b<c es incorrecto, debe ser (a<b) Y (b<c).

3) Stacks de Operandos y Operadores

Pasemos ahora a revisar la composición de las pilas o stacks para procesar expresiones.

Un operando puede representar a una variable, a un temporal y también a una constante de tipo real o una literal numérica, que al pasarse al código será en forma de literal numérica.

Cuando se tenga una variable o un temporal en el operando es importante determinar que clase de estructura se representa o al menos saber si es real o texto.

Así pues, se decidió diseñar al stack de operandos como sigue:

```
VAR STOPNDO : PACKED ARRAY [ O..TSTOPNDO]
              OF TOPNDO;
```

donde TSTONDO es una constante y TOPNDO:

```
TYPE
       TTIPO = 0..15;
       TOPNDO = PACKED RECORD
                  CASE TIPO : TTIPO OF
                   1 : (LOCR : REAL);
                   0,2,3,4,5,6,7,8,9,10,11:
                     (LOC, NIV : INTEGER)
                END;
```

El tipo que contiene el stack de operandos (TIPO : TTIPO) es distinto al que traen los descriptores. Este nuevo tipo al que voy a llamarle tipo del operando indica que clase de estructura representa el operando.

Hasta el momento se han especificado nueve tipos de estructuras de datos, mediante los cuales, se puede representar cualquiera de las variables cuya declaración ya ha sido implantada, independientemente de que requieran de una o varias de esas estructuras de datos para su representación.

A cada una de las estructuras de datos referidas, le corresponde un valor en el campo TIPO del operando y a continuación se muestra por cada valor, el objeto que representa:

1. Se representa una literal numérica o el valor de una constante tipo real. El número real referido se guarda en el campo LOCR. Esta estructura es totalmente distinta a las demás, pues no ocupará espacio en la memoria del programa objeto, sino que pasará como una literal numérica.

Todas las demás estructuras dispondrán de dos campos, en lugar del campo LOCR. Ellos son LOC y NIV. El segundo campo indica a que bloque o nivel lexicográfico pertenece la estructura y LOC es el desplazamiento relativo al inicio del bloque en donde estará situada la estructura en la ejecución. En caso de que el operando represente el resultado de una operación, es decir,

un operando temporal, el campo LOC contendrá un -1. Esto se de be a que el temporal vivirá en el tope del stack de memoria (TOPEMEM), ya que el espacio ocupado por él, puede ser muy gran de y hasta impredecible en compilación. Así pues no basta tener un espacio fijo por bloque para almacenar temporales. El tipo sólo servirá para indicar si se tiene un real o un texto, pues en compilación no siempre se puede determinar la forma de la estructura resultante (escalar o arreglo). El campo NIV no se empleará. Más tarde se verá como se determina en ejecución la verdadera localidad y clase de la estructura resultante.

Como puede verse, cuando un operando representa una variable, la información dispuesta es suficiente para generar código, que direccione correctamente a dicha variable.

El resto de las estructuras son:

2. Variable escalar de tipo real.
3. Variable escalar de tipo texto o cadena.
4. Arreglo de tipo real. En estas estructuras se desconoce el número de dimensiones. La validación de que el número de di mensiones en dos operandos a operar sean iguales, se deja a ejecución. Inclusive, en un operando temporal no se puede saber si representa a un escalar o a un arreglo, de tal manera que esta clase de validación también se deja al programa objeto.
5. Arreglo de tipo texto.
6. Gráfica.
7. Tabla.
8. Histograma.

   Estas tres últimas estructuras no pueden participar en una operación, sin embargo, una tabla se puede asignar a otra ta bla, en una sola proposición, sin tener que hacerlo campo por campo. Es por ello que estas tres variables tienen sendas representaciones, aunque sólo se permita que aparezcan aisladas, en una expresión.

9. Cuerpo de Gráfica. Esta estructura es muy especial, pues contiene la imagen de una gráfica, punto por punto. La única operación permitida para este tipo de estructura es la suma, que consiste en unir dos gráficas, respetando los símbolos originales de graficación.

   Esta operación aún no se implanta, pues tampoco se ha implantado la construcción del cuerpo de una gráfica. Como ya antes se mencionó, se consideró conveniente instaurar esta parte del lenguaje hasta tener constituida la entrada-salida del lenguaje.

Los tipos 0, 10 y 11 no se emplean, se han dejado para posibles futuros usos.

La rutina empleada para meter un elemento al stack de operandos es MOPNDO y la rutina que saca un elemento es SOPNDO.

Pasemos ahora a ver el stack de operadores. Este se define como sigue:

```
VAR STOPDOR : ARRAY [ 0..TSTOPDOR] OF
                    TOPDOR;
TYPE TOPDOR =INTERGER;
```

Donde TSTOPDOR es una constante.

Cada operador en el lenguaje tiene asignado un código para representarlo en el stack. Ellos se enumeran a continuación, de acuerdo con su precedencia, comenzando con los de menor precedencia.

Precedencia	Operador	Código y Nombre
1	< =	1. Menor o igual
1	< >	2. Distinto
1	<	3. Menor estrictamente
1	=	4. Igual
1	> =	5. Mayor o igual
1	>	6. Mayor estrictamente
2	+	7. Suma
2	-	8. Resta
2	O	9. O - inclusivo
3	-	16. Menos unario
3	+	15 y 17. Más unario (17,cuando el operador unario es vacío)
4	*	10. Multiplicación
4	/	11. División
4	> <	12. Multiplicación de matrices

Precedencia	Operador	Código y Nombre
4	Y	13. Conjunción
5	**	14. Potencia
6	@	19. Inversa de matriz
6	´	20. Traspuesta de matriz
7	NO	18. Negación
8	!<EXP>!	21. Determinante de la matriz <EXP>

Si se desea conocer a fondo el significado y comportamiento de cada operador, se recomienda leer a García [G1], la sección dedicada a expresiones en el tercer capítulo.

Las rutinas para administrar el stack de operadores son, para meter un elemento al stack: MOPDOR, y para sacar un elemento del stack: SOPDOR.

Pasemos a la siguiente sección, a examinar como el programa objeto lleva a cabo la evaluación de las operaciones generadas y como maneja a los operandos temporales.

4. Evaluación de Operaciones en el Programa Objeto

Todas las operaciones definidas en el lenguaje y que se acaban de revisar, se efectúan por medio de llamados a las rutinas OPERBIN y OPERUNA. Estas rutinas viven en el programa objeto como funciones de biblioteca y se encargan de realizar las operaciones binarias y unarias respectivamente.

Los encabezados de las declaraciones de esas rutinas se presentan a continuación:

```
PROCEDURE     OPERBIN (OPER : TIPOPER; LC1 : REAL;
              TP1 : INTEGER; LC2 : REAL; TP2 : INTEGER);

PROCEDURE     OPERUNA (OPER : TIPOPER; LC1 : REAL;
              TP1 : INTEGER);
```

El tipo TIPOPER comprende los nombres de todas las operaciones permitidas en el lenguaje y OPER es la variable que especifica la operación a ejecutar.

También se describe al, o a los operandos que intervienen en la operación. Se especifica el tipo de la estructura (TP1, TP2)*

* En las operaciones binarias TP1 y LC1 representan al operando izquierdo, mientras que TP2 y LC2 lo hacen para el derecho.

y la localidad o valor de la estructura (LC1, LC2).

Es importante hacer mención, que el tipo aquí referido, es el mismo tipo que se emplea para representar a los operandos en compilación.

Los datos que intervienen en una operación son de índole heterogénea, pueden ser reales o arreglos, de tal manera, que para poder describir claramente a cada estructura, se tuvo que crear una codificación especial. Un problema parecido ocurre en compilación. Como los objetos referidos en ambos procesos son los mismos, se decidió tomar la misma codificación para ambos procedimientos.

Si el operando corresponde a una literal numérica, el tipo contiene un 1 y el campo de localidad (LC1, LC2) posee el valor del número real respectivo. Si lo que se tiene, es un temporal, entonces el campo de localidad vale -1. Para determinar la verdadera localidad y tipo de la estructura representada se hace uso de un stack de operandos parecido al que existe en compilación. Más tarde se verá como es y como funciona dicho stack. Si no se está en ninguna de esas dos alternativas, entonces el operando hace referencia a una variable. El tipo contiene la clase de estructura y la localidad, al desplazamiento relativo al bloque, en donde vive la variable. El nivel o bloque al que pertenece la variable se maneja en una variable global (NIVELUNO y NIVELDOS para los operandos izquierdo y derecho respectivamente), a fin de no desperdiciar un parámetro, cuando el operando corresponde a un temporal o a una literal numérica. En otras palabras, se genera una asignación a la variable NIVELUNO o NIVELDOS. Con estos elementos y utilizando el arreglo DISPLAY (registros de despliegue), se puede determinar la dirección exacta de la estructura, en el stack de memoria.

Habiendo determinado perfectamente a los operandos de la operación, se procede a validar que sean compatibles entre sí y con la operación (que los operandos sean del mismo tipo y que la operación pueda aplicarse, por ejemplo, sólo se puede trasponer un arreglo bidimensional y las cadenas sólo pueden concatenarse,

"sumarse"). Si no hay anomalías, se empieza a operar y a construir el resultado, el cual se coloca a partir del tope del stack (TOPEMEM).

Finalmente se determina el tipo de la estructura resultante y junto con su dirección en memoria (TOPEMEM), se construye el operando temporal que se coloca en el stack de operandos temporales. Además se actualiza el tope del stack, mediante la rutina LONG, que recibe la descripción del temporal y entrega su longitud en celdas de memoria.

Ahora, pasemos a ver el stack de operandos en ejecución.

5. Stack de Descriptores de Operandos en el Programa Objeto

La pila de operandos en el código se utiliza para guardar las descripciones de los operandos temporales que van apareciendo al procesar una expresión.

Dada la filosofía del lenguaje, es imposible en compilación determinar la longitud de cada resultado, de tal manera, que se vuelve imprescindible tener en ejecución un mecanismo que describa perfectamente a los temporales. Veamos un ejemplo que se encuentre en esta situación:

Sean I, J, K, L variables escalares y A y B, dos arreglos tipo columna.

La expresión A[I _ J]><(B[K _ L])' puede resultar en un escalar o en una matriz cuyo número de elementos será dado por (J - I + 1) * (L - K + 1). Así el tipo y localidad del temporal sólo se podrán determinar hasta ejecución y por eso se incluyó al stack de descriptores de operandos temporales en ejecución, como parte del código para evaluar expresiones.

El stack está diseñado como sigue:

```
VAR STOPNDO : ARRAY[0..TAMSTOPNDO] OF TOPNDO;
TYPE TOPNDO = RECORD
                CLOC : INTEGER;
                CTIPO: INTEGER;
              END;
```

TAMSTOPNDO es una constante.

Cada elemento del stack contiene dos valores, el tipo de la estructura (CTIPO) y la dirección exacta en memoria de dicha estructura (CLOC). Mediante esta información se puede representar cualquier estructura almacenada en el stack de memoria.

Además se dispone de dos rutinas: METE que carga un elemento al stack y SACA que obtiene el operando en el tope de la pila.

Cada vez que se efectúa una operación, la descripción del operando resultante se mete al stack y cada vez que un operando fuente de una operación represente a un temporal, se saca un elemento del stack.

Como puede verse, el manejo del stack es muy sencillo. Sólo hay que tener cuidado cuando los dos operandos de una operación binaria corresponden a temporales. En este caso hay que sacar primero al elemento correspondiente al operando derecho, dado que las expresiones se evaluan de izquierda a derecha y los operadores con asociatividad, la tienen definida de izquierda a derecha.

Finalmente veamos un ejemplo de como se comporta el stack al evaluar una expresión:

Sea la expresión : (A+B)*(C+D)<=(E+F)*(G+H)

El stack pasará por las siguientes configuraciones:

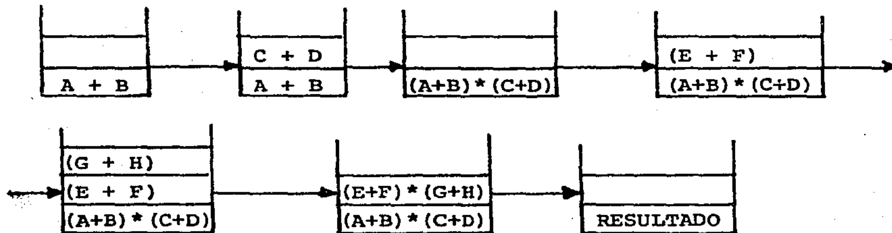


Figura 6.14 Stack de Descripciones de Operandos en Ejecución.

6. Arreglos Explícitos

Los arreglos explícitos son literales o constantes numéricas en forma de arreglo, que pueden aparecer en las expresiones del lenguaje. Para conocer más a fondo este tipo de estructuras, como se construyen e interpretan, es conveniente acudir a García [G1].

A la definición gramatical original de arreglos explícitos, se le agregó los llamados semánticos que los procesan. Dicha definición se presenta a continuación:

```
<Arreglo Explícito> :: = "[[" <Límites Exp> ":"
                             <Lista de Exp Rep> "]]" <5037>
<Límites Exp> :: = <5036> <Num o Cons> <R7>
<R7> :: = "," <Num o Cons> <R7> | e
<Lista de Exp Rep> :: = <Exp Rep> <R8>
<R8> :: = "," <Exp Rep> <R8> | e
<Exp Rep> :: = <5038> <Num o Cons> "("
               <Lista de Exp Rep> ")" <5039> |
               <Exp> <5040>
```

Hay una pequeña mejora en esta definición con respecto a la versión original. Antes, en los límites del arreglo sólo se permitian números, ahora se pueden usar número y constantes reales.

Los arreglos explícitos realmente son arreglos que pueden ser de tipo real o de tipo texto, que aparecen y desaparecen en el programa objeto durante la evaluación de una expresión y se inicializan con una serie de valores que no cambiarán mientras exista el arreglo.

Como en el lenguaje no hay arreglos de tipo heterogéneo, es decir, que incluyan reales y textos, todas las expresiones que inicien al arreglo explícito deben ser del mismo tipo, reales o textos.

Así como en la declaración de arreglos se especifica el número de dimensiones y el tamaño de cada dimensión, en la definición del arreglo explícito, también se especifica lo mismo.

Para poder representar al arreglo explícito en memoria se requiere en primer lugar, apartar espacio y luego asignar la descripción del arreglo. Todo este trabajo corre a cargo de la rutina BXARREXP, que coloca a la estructura a partir de donde esté situado el tope del stack de memoria (TOPEMEM).

Para inicializar los elementos del arreglo explícito en el programa objeto se emplea una estructura de datos que es generada durante la compilación. Dicha estructura contiene una representación de las expresiones y ciclos de repetición que componen a la lista de expresiones repetitivas (<Lista de Exp Rep>) y consiste de un arreglo lineal de enteros llamado ARREXP, cuyos elementos van del cero hasta el ARREXPTAM, que es una constante entera. El elemento cero contiene el número de celdas empleadas para la representación y el resto del arreglo contiene la siguiente codificación:

Sea X el valor de una celda del arreglo.

Si X es positivo, representa una repetición de expresiones, que abarca desde el siguiente elemento hasta el elemento cuyo valor sea 10,000 y que representa el fín del ciclo. Cabe indicar que los ciclos de repetición se pueden anidar tantos niveles, como capacidad tenga ARREXP para almacenar a toda la representación.

Si X es menor o igual a cero, representa a una expresión y es el negativo de la posición que tiene dicha expresión en el stack de operandos (STOPNDO).

Esta codificación ocasiona que las variables y las literales numéricas deben tener una descripción en el stack de operandos a pesar de no ser temporales y por tanto que las literales numéricas vivan en la memoria. Se decidió construir así a la representación, porque la repetición de expresiones puede ser teóricamente, cualquier valor positivo y porque si se quisiera ubicar a las literales numéricas en el mismo ARREXP, este tendría que ser de reales y no de enteros y se tendría que tener un dato adicional que indicara que tipo de elemento hay en cada celda de ARREXP.

Veamos un ejemplo del uso de ARREXP.

Sea el arreglo explícito:

[[5,5 : 3((A+B)*C, 2(10,ARR,CTE1))]]

Donde A, B y C son escalares reales, ARR es un arreglo lineal de dos elementos y CTE1 es una constante real.

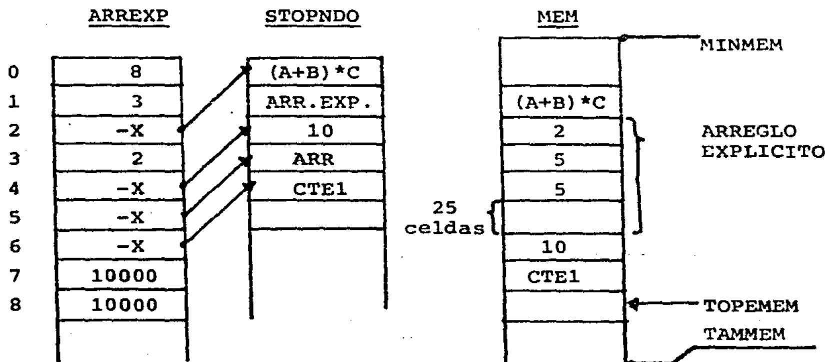


Figura 6.15 Representación de un Arreglo Explícito en el Programa Objeto.

El elemento cero de ARREXP indica que hay ocho elementos para representar al arreglo explícito. En éste hay dos ciclos de repetición de expresiones, el primero indicado por los elementos 1 y 8 y el más interno, formado por los elementos 3 y 7. El resto de los componentes son negativos y son los negativos de los apuntadores al stack de operandos en donde viven las descripciones de cada expresión. Por último se muestra el contenido del stack de memoria en donde vive el temporal (A+B)*C, el arreglo explícito y las literales numéricas. Todos ellos viven como temporales.

Como puede verse en el ejemplo, la representación de ARREXP en STPONDO y el espacio ocupado por él en MEM están después de la representación de la primera expresión en el arreglo explícito. Esto se debe a que sólo se puede determinar el espacio que va a ocupar el arreglo explícito, hasta conocer su tipo (real o texto) y éste se determina, sólo hasta conocer el tipo de la

primera expresión.

Se trata de que arreglo explícito viva en MEM antes que los temporales de las expresiones que lo constituyen, a fín de que cuando se haya constituido completamente al arreglo explícito, todos los temporales empleados se puedan eliminar y no haya tanto espacio desperdiciado en la memoria.

Las rutinas que emplea el programa objeto para construir un arreglo explícito son: BXARREXP y DXARREXP que apartan el espacio en memoria para almacenar al arreglo y asignan el número y tamaño de dimensiones. AXARREXP, que almacena a una literal o constante numérica en la memoria y en el stack de operandos. CXARREXP, que coloca la descripción de una variable en el stack de operandos y finalmente la rutina CONSARREXP que toma a ARREXP y empleando recursividad inicializa el contenido del arreglo explícito. Al terminar de efectuarse la inicialización, se actualiza el tope al stack de memoria y también se actualiza el stack de operandos, eliminando a los elementos siguientes al operando del arreglo explícito y también al que está inmediatamente antes de él.

En compilación también se dispone de una estructura para describir al arreglo. Consiste en un arreglo lineal de enteros, que también se llama ARREXP y que va de -3 hasta MAXDIMSIM (el número máximo de dimensiones que puede tener un arreglo en el lenguaje). El uso de cada elemento de ARREXP se explica a continuación:

ELEMENTOS	FUNCION
1...MAXDIMSIM	Contienen el tamaño de cada dimensión del arreglo explícito
0	Número de dimensiones
-1	Número de elementos que contiene el arreglo explícito
-2	Tipo del arreglo: 0.- Aún sin tipo 1.- Real 2.- Texto
-3	Indice a la última celda del arreglo ARREXP del programa objeto, a la que se ha generado una asignación

Los llamados semánticos que aparecen en la definición gramatical del arreglo explícito se encargan de hacer lo siguiente:

5036.- Determina las dimensiones del arreglo y construye parcialmente al arreglo ARREXP de compilación.

5040.- Analiza una expresión y genera un llamado a AXARREP, a CXARREXP o simplemente genera una asignación a ARREXP. Cuando se procesa a la primera expresión del arreglo, se determina el tipo del mismo y se generan llamados a BXARREXP y DXARREXP para que construyan la estructura de tipo arreglo y es aquí donde más se emplea a la tabla ARREXP del compilador.

5038 y 5039.- Procesan un ciclo de repetición de expresiones. Validan que la repetición sea mayor a cero (5038) y generan la asignación a ARREXP correspondiente.

5037. Genera un llamado a la rutina CONSARREXP.

Por último es importante mencionar que no se permite tener un arreglo explícito en la lista de expresiones de otro arreglo explícito, aunque si se permite cualquier otro tipo de expresión. Esto se prohibió porque se puede evitar tener un arreglo explícito dentro de otro, escribiendo la lista de elementos de aquel en la lista de expresiones de éste y porque se complica mucho la compilación y construcción del arreglo en ejecución al tener que implantar estructuras que soporten recursividad, como ocurre en la selección de arreglos (obtención de subarreglos a partir de arreglos) que es el siguiente punto a revisar.

7. Expresión Selectiva de Arreglos

Se llama así a la expresión que permite obtener de un arreglo n-dimensional de tipo real o texto, una estructura de tipo escalar o una estructura de tipo arreglo, con n o menos dimensiones, por medio de seleccionar o sumar elementos de aquel y también permite asignar una estructura de tipo escalar o de tipo arreglo, a un conjunto de elementos de otra estructura tipo arreglo, con igual o mayor número de dimensiones.

La definición gramatical de la expresión selectiva de arreglo se presenta a continuación.

<Exp Selec> :: = <Exp Selec de Arr> | e
<Exp Selec de Arr> :: = "[" <5041> <Lista de Dim> "]"
<Lista de Dim> :: = <Elem de Dim y Pto.> <R12>
<R12> :: = "," <Elem de Dim y Pto.> <R12> | e
<Elem de Dim y Pto.> :: = "." | <Elem de Dim> | e
<Elem de Dim> :: = <Grupo de Elem> <R13>
<R13> :: = "&" <Grupo de Elem> <R13> | e
<Grupo de Elem> :: = <Exp> <5034> <Intervalo>
<Intervalo> :: = "_" <Exp> <5034> | e

A la versión original de la gramática sólo se le agregaron tres llamados semánticos. Los dos llamados 5034 se utilizan para que el compilador se entere que en ese punto finaliza la parte del árbol sintáctico, correspondiente a una expresión.

La gran utilidad del llamado 5034 consiste en poder procesar una expresión que sea llamado recursivamente desde otra expresión, con la que tenga cierta independencia, como es el caso de las expresiones que definen a los intervalos de elementos, dentro de la expresión selectiva de arreglo. Por ejemplo, si defino a <Exp 1> como A[<Exp 2> _ <Exp 3>]. Una expresión que representa el rango de elementos: {<Exp 2 ,..., <Exp 3>} del arreglo A. Para procesar a <Exp 1> se llama a la rutina PROCEXP. Esta rutina se llama recursivamente a si misma para procesar a <Exp 2> y a <Exp 3>. Para finalizar el análisis de <Exp 2> o <Exp 3> y regresar a <Exp 1>, se utiliza el llamado 5034, que se comporta como una bandera que indica el fín de la expresión en proceso.

El llamado 5041, se emplea para procesar a toda la expresión selectiva de arreglo, mediante la rutina EXPSELEC.

Como puede observarse en la gramática, la expresión selectiva puede no aparecer, pues también se puede tomar un arreglo en forma íntegra, para participar en una operación o para asignarlo a otro.

Los elementos a seleccionar se especifican por cada dimensión del arreglo, mediante una expresión. Estas expresiones pueden contener a los siguientes operadores:

VACIÓ.- Si la expresión es vacia, se toman todos los elementos de la dimensión. Por ejemplo, si A es un arreglo bidimensional entonces A[,] representa lo mismo que A.

.(Punto).- Se suman todos los elementos de la dimensión.

&.- Se utiliza para seleccionar dos o más rangos o intervalos de elementos, en una dimensión.

_(Subrayado).- Marca un intervalo o rango de elementos en una dimensión.

Ver un ejemplo en la figura 6.16 .

Cuando se aplica la selección, la estructura resultante puede quedar con el mismo número de dimensiones, con menos o ser un escalar. Para conocer más a fondo la filosofía de la expresión selectiva de arreglo revisar a García [G1].

La expresión selectiva también se puede aplicar a una variable a la que se asigne una expresión. En este caso la expresión selectiva indica a que elemento o a que rangos de elementos se les asignará la expresión, del lado derecho de la proposición de asignación.

Cuando la expresión selectiva se aplica en una asignación o a una expresión de tipo texto, el operador punto, como es de esperarse queda vedado.
Al igual que en la construcción de un arreglo explícito, al efectuar el programa objeto una selección de arreglo se auxilia de una estructura de datos, que es generada desde la compilación y que en este caso se llama SELEC. Esta estructura es un arreglo lineal de enteros, que va de -1 hasta TAMSELEC (una constante).

SELEC representa a la expresión selectiva por medio de los elementos que designan los rangos a substraer.

Mientras en el elemento cero de SELEC se coloca el número de dimensiones que denota la expresión selectiva, en el resto del

arreglo se codifican, por cada dimensión, los rangos de elementos pedidos.

En el primer elemento de cada dimensión se coloca el número de intervalos pedidos y cada intervalo se codifica en uno o dos valores, como sigue. Los rangos de los elementos de un intervalo se representan con X.

Si se tiene la suma de todos los elementos, la codificación consiste de un sólo valor: -1001 (X= - 1001). (Ver la figura 6.17).

Si el operador empleado es el vacio, se abarca integramente a toda la dimensión. También se ocupa una sola celda, con el valor -1002. (X2 = -1002).

Si lo que se tiene es un elemento o un rango de elementos, entonces éstos se codifican así:

El elemento es una literal real. Como es un índice a arreglo, se verifica que sea mayor o igual a 1, se redondea y se asigna a SELEC. (X > = 1).

El elemento es una variable. Como dicha variable se empleará como un índice a arreglo, se verifica que sea un escalar real. El elemento se codifica: -(DISPLAY[NIVEL] + LOC + 1003), se toma el negativo de la suma de 1003 y de la dirección en memoria (que es mayor a cero) de la variable. (X < = -1003). Toda esta expresión es generada al código, pues en compilación sólo se conoce el nivel y desplazamiento de la variable.

Si el elemento corresponde a una expresión, entonces posee una descripción en el stack de operandos (STOPNDO). Se toma el negativo de esa posición, que corresponde a la expresión -(ISTOPNDO -1).

ISTOPNDO es el apuntador al stack de operandos y siempre señala al primer lugar libre del stack. La expresión anterior si puede ser generada en compilación, pero hay que emitirla después de procesar a la expresión electora. Sin embargo, con el fín de grabar menos código se decidió mejor emitir a - ISTOPNDO. La única complicación es que en la decodificación de SELEC, hay que tomar en cuenta dicha convención. (-1000 < = X < = 0).

En caso de que el intervalo en la expresión consista de un sólo elemento, por ejemplo A[I], el segundo intervalo de la codificación tendrá el valor de -1000, a fín de indicar que es idéntico al primero. Esta representación equivale a la expresión A[I_I]. (X = -1000).

Como puede observarse en los valores de X, hay seis rango de valores independientes entre sí, para representar a todas las clases posibles de elementos electores evitando confusiones.

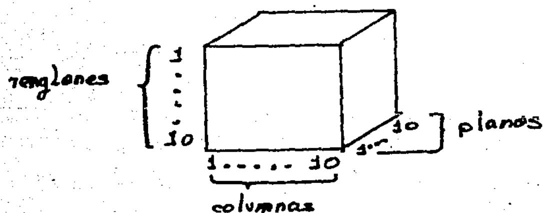
Veamos un ejemplo del empleo de SELEC:

Sea A un arreglo en forma de cubo cuya declaración está dada así: A[1:10, 1:10, 1:10].

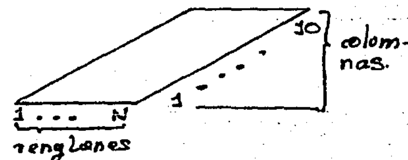
Sean b, c y d variables escalares reales.

Sea la expresión A[., 4_6 & 5 & b_c*d,]. La estructura resultante es un arreglo de dos dimensiones constituído por la suma de todos los renglones de A, algunas columnas y todos los planos. Enseguida se muestra gráficamente al arreglo A y a la expresión resultante.

A[1:10, 1:10, 1:10]



A[.,4_6&5&b_c*d,]



Los N renglones se formaron a partir de las columnas 4_6,5 y b_c*d.

Figura 6.16 Ejemplo de una Aplicación de la Expresión Selectiva de Arreglo.

Veamos enseguida el contenido del arreglo SELEC para este ejemplo.

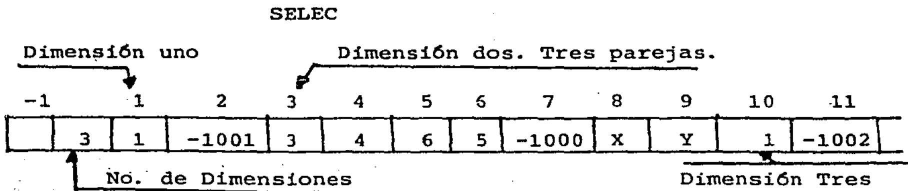


Figura 6.17 Representación de una expresión Selectiva de Arreglo en SELEC.

El contenido de SELEC indica que corresponde a un arreglo que debe ser de tres dimensiones. La primera dimensión contiene un sólo rango y éste consiste del valor -1001 que se interpreta como la suma de todos los renglones. En la segunda dimensión hay tres rangos: de la columna 4 a la 6, de la 5 a la 5 y el otro rango está dado por una variable, cuya dirección en memoria es -X + 1003 y por una expresión, cuya descripción vive en el stack de operandos, en la posición: -Y-1. Por último, en la tercera dimensión hay un sólo rango, el designado por -1002, que corresponde al operador vacío.

Como puede examinarse, el orden en que se escojan los elementos en una dimensión es arbitrario y un mismo elemento puede seleccionarse más de una vez, tal y como ocurre en la segunda dimensión.

Hay que dejar muy en claro, que conforme se designen los rangos de elementos en la expresión selectiva, es como se irán tomando o como se irán asignando, según el caso.

Si en una asignación se repite un elemento en una dimensión, a diferencia de lo que ocurre en una expresión, aquí se considerará sólo a la última referencia de dicho elemento.

V.gr.: A[1&1] : = [[2:10,20]] equivale a A[1]: = 20;
y [[2:10, 20]][1&1&1] equivale a [[3:10, 10, 10]].

La celda -1 de SELEC sólo se utiliza en algunos casos, cuando haya direccionamiento indirecto; es decir que dentro de una expresión selectiva aparezca otra selección. En estos casos el

contenido de SELEC es guardado en memoria y la dirección correspondiente es guardada en el lugar -1. Al terminar de realizar una selección interna, el contenido de la selección que inmediatamente la contiene es restaurado, por medio de la dirección en la celda -1.

Las rutinas que se encargan de almacenar y restaurar el contenido de SELEC son GRABAES y LEEES respectivamente.

Ya que la estructura SELEC se haya terminado de construir, es tomada por la rutina DECODSELEC que la decodifica y genera otras estructuras que serán tomadas por la rutina RUTSEL, que es quien se encargará de obtener el subarreglo o de asignar una estructura al subarreglo. Cabe recordar que una expresión selectiva también puede estar del lado izquierdo de una asignación para indicar en que elementos de un arreglo se asignará el resultado del lado derecho de la asignación.

La primera estructura adicional es SELECB, que es un arreglo paralelo a SELEC, que contiene los valores reales de los intervalos.

Para el ejemplo mostrado con anterioridad, se tendría a SELECB como:

Si b = 1, c = 2, d = 2:

SELECB

1	2	3	4	5	6	7	8	9	10	11	
1	-10	3	4	6	5	5	1	4	1	-10	

Figura 6.18 Representación de una Expresión Selectiva en SELECB.

En SELECB los valores de -1001 y -1002 son sustituidos por el negativo del tamaño de la dimensión. Son negativos para indicar que representan intervalos especiales, pues consisten de un sólo elemento y no de dos. Para poder diferenciar los valores negativos, según representen operadores punto u operadores vacío, se hace uso de otras estructuras: LOCSEL y LOCRES. Fi-

nalmente, en la segunda dimensión las columnas a tomar son: de la 4 a la 6, la 5 y de la 1 a la 4, exactamente en este orden y en total ocho columnas.

El papel de las otras estructuras mencionadas es:

LOCSEL.- Un arreglo lineal que apunta a donde comienza cada dimensión en SELEC y SELECB.

LOCRES.- Un arreglo lineal que apunta a LOCSEL, a aquellas dimensiones cuyo número de elementos seleccionados es mayor a uno.
Mediante este arreglo se puede determinar si un elemento de SELECB corresponde a un operador punto o a un operador vacío y también se puede determinar cuantas dimensiones tendrá el subarreglo resultante y a cuales dimensiones corresponden en el arreglo fuente o en el caso de una asignación, cuantas dimensiones debe tener la expresión a asignar y a cuales dimensiones corresponden en el arreglo afectado por la selección.

Para el ejemplo discutido, el contenido de estas estructuras será:

LOCSEL

1	2	3	4		MAXDIM
1	3	10	0		

LOCRES

1	2	3		MAXDIM
2	3	0		

MAXDIM = Máximo número de dimensiones que puede haber en un arreglo.

Figura 6.19 Representación de una Expresión Selectiva en LOCSEL y LOCRES.

LOCSEL muestra que la dimensión uno define a partir de la celda 1

en SELEC y SELECB y los otros elementos indican lo correspondien te para las siguientes dimensiones. LOCRES indica que el subarreglo consistirá de dos dimensiones. La primera corresponde a la segunda del arreglo seleccionado y la segunda, a la tercera de ese.

Ya que se ha decodificado el contenido de SELEC y se hayan constituido los arreglos SELECB, LOCSEL y LOCRES se pasa el control a la rutina RUTSEL, que se encarga de llevar a cabo la obtención del subarreglo o la asignación al subarreglo.

RUTSEL recorre todos los intervalos de todas las dimensiones en SELECB, manipulando cada uno de los elementos involucrados en ese conjunto de rangos. Si se tiene la obtención de un subarreglo, los elementos escogidos, se van dejando uno por uno, en el arreglo destino. Si se tiene una asignación, entonces a cada elemento en el subarreglo, se le va asignando el elemento correspondiente del arreglo fuente.

Para saber que elemento en el arreglo sin selección le corresponde a cada elemento del arreglo seleccionado (el arreglo afectado por la expresión selectiva), se diseñó un procedimiento que se expone a continuación.

Mediante LOCRES se puede determinar el número de dimensiones que tendrá o que deba tener el arreglo sin selección y también se puede determinar para cada dimensión de éste, cual es su correspondiente en el arreglo seleccionado.

Por cada dimensión en SELEC que esté referenciada por LOCRES, se tiene un contador que comienza en uno y sirve para contabilizar a todos los elementos seleccionados en esa dimensión. Todos estos contadores en conjunto, representan los índices de todos los elementos del arreglo sin selección y están contenidos en el arreglo CB. Así el elemento (i1, i2,...,in) del arreglo con selección, le corresponde el elemento (j1, j2,...jm) del arreglo sin selección. Tomando el ejemplo que se ha venido desarrollando, al elemento (1,4,1) del arreglo A, le corresponde el elemento (1,1) del arreglo resultante, ya que la columna 4 del arreglo A le corresponde al renglón 1 del arreglo sin selec

ción y el plano 1 de A le corresponde a la columna 1 del otro arreglo.

Los índices al arreglo con selección (i1,...,in) viven en el arreglo C y los índices al arreglo sin selección (j1,...,jm) viven en el arreglo CB.

Cuando se utiliza el operador punto, la dimensión que lo contenga no tiene representación en LOCRES, pues es una dimensión que se pierde. Entonces hay muchos elementos del arreglo con selección, que tendrán el mismo elemento representante en el otro arreglo, tantos como haya en esa dimensión. Para facilitar el proceso de selección, como el operador punto no se puede emplear en una asignación, cuando se va a obtener un subarreglo, primero se inicializa a la estructura resultante en ceros y luego se acumulan los elementos tomados, conforme se vayan obteniendo. Volviendo al ejemplo presentado, todos los elementos (i,4,1) i=1,...,10 del arreglo A corresponden al elemento (1,1) del otro arreglo. Entonces $(1,1) = \sum_{i=1,...,10} (i,4,1)$.

Cuando en una asignación sobran elementos a asignar, ya no se toman en cuenta y si hay elementos a los que ya no hay que se les asigne, se dejan inalterados.

La rutina DECODSELEC, que decodifica el contenido de SELEC, utiliza mucho el stack de operandos (debido a los temporales a que hace referencia SELEC), por lo que se tiene que hacer un manejo, un tanto artificial de él, en dos ocasiones. Primero, cuando participan temporales en la expresión selectiva. Al terminar de procesar la selección, hay que eliminar los temporales del stack, para ello se toma la posición del primer temporal que esté en SELEC y se actualiza a ISTOPNDO (tope del stack de operandos). Cuando se va a obtener un subarreglo y se va obtener a partir de un temporal, éste vive en STOPNDO, debajo de todos los temporales que pertenecen a la expresión selectiva. Como el temporal del arreglo con selección debe estar en STOPNO, inmediatamente antes que el descriptor del primer temporal en la selección, a partir de la información en SELEC, se localiza a aquel y se copia al tope del stack, para

que la ejecución continué normalmente. Al finalizar la selección, hay que eliminar ese operando que representa al arreglo con selección y que aún está presente en el stack STOPNDO.

Finalmente, al obtener un subarreglo, se inserta su descriptor al stack de operandos (STOPNDO).

En el planteamiento original del lenguaje se ideó que el orden en que se seleccionarán los elementos sería en forma ascendente, es decir en el orden original de la dimensión y no en el designado por la expresión, como ocurre aquí y además, que los elementos seleccionados no pudieran repetirse, al contrario de lo que ocurre aquí. Estos cambios se tomaron porque se consideró que volverían más flexible y poderosa a la expresión selectiva de arreglo.

Cuando el monitor del analizador semántico se encuentra activado, por medio de la opción de compilación $$D+, se emite una reseña detallada del procesamiento de una expresión, mostrándose entre otras cosas, los cambios que ocurran en los stacks de operandos y operadores, el momento de generar una operación binaria o unaria y los descriptores de las variables participantes.

En la figura 6.20 se presenta un programa que ejemplifica la utilización de las expresiones y también el empleo de la proposición de asignación, de la que trata la siguiente sección. También se presenta parte de la reseña semántica de la compilación, el programa objeto generado y parte del listado que emite la ejecución del programa objeto.

Con el fín de poder apreciar las ventajas de las expresiones que pueden escribirse en el lenguaje estadístico, se presenta en la figura 6.20.A el mismo programa de la figura 6.20 pero escrito en PASCAL.

Al revisar el programa escrito en PASCAL hay algunos puntos que quisiera explicar:

Se inicializan los arreglos EDADES y SEGEDADES porque así se hace en el lenguaje estadístico y en el caso específico de SEGEDADES, si esta variable no se inicializara, habría basura

```
                % UTILIZACION  DE EXPRESIONES:
$$D+ ←————————— Se Prende Monitor de Semántica.
INICIO
   CONSTANTE
      TAMEDADES = 8;
   ARREGLO
      EDADES, SEGEDADES [ ]:2. ]:TAMEDADES ];
   REAL
      ED, SUBTOTAL, EDADMUJERES;

   % INICIALIZACION DEL ARREGLO EDADES :
   EDADES:= [[ 2, TAMEDADES : 3 ( 1, 7, 3 ( PI, E, 15 ), 11 ) ]];

   % SUMA DE LAS EDADES DE LAS MUJERES ENTRE 5 Y 90 AÑOS ;
   EDADMUJERES:= (( (EDADES[1, ]>=5) Y (EDADES[1, ]<=90) )*EDADES[1, ] ) [ . ];

   % INICIALIZACION DE SEGEDADES ;
   SEGEDADES[ 2, 4_8 & 1_3 ]:=
      [[ TAMEDADES : EDADES[ ], 62_66 ] ]];
←
FIN;
```

Figura 6.20 Programa que ejemplifica el uso de expresiones y asignaciones.

en su primer renglón.

Se incluyó la rutina ASIGNA a fin de facilitar las asignaciones a EDADES y para que al igual que en el otro programa, se asignen tantos elementos como el valor de TAMEDADES lo permita.

En la asignación a SEGEDADES se emplea el arreglo auxiliar ARRAUX porque en el programa del lenguaje estadístico se utilizó un arreglo explícito que vendría a equivaler a ARRAUX.

Al comparar los dos programas, el de PASCAL contiene mucho más variables declaradas y es mucho más grande. Las asignaciones a EDADES y a SEGEDADES son más complicadas en PASCAL. La inicialización de variables en PASCAL tiene que hacerse como si fuera parte del programa y no implícito al lenguaje. Sólo en el caso de la asignación a EDADMUJERES, la expresión del lenguaje estadístico es más complicada que el procedimiento que hay en PASCAL.

Como en la ejecución del programa del lenguaje estadístico se emite mucho monitoreo, resulta ésta más lenta que la del programa en PASCAL.

Finalmente, en las figuras 6.23 y 6.23.A se muestra la salida de los dos programas.

Figura 6.20.A Programa en PASCAL que emula al programa de la figura 6.20 .

```
(* PROGRAMA DE COMPARACION ENTRE PASCAL Y EL LENGUAJE ESTADISTICO *)
PROGRAM EMULA;

  CONST TAMEDADES=8;
  TYPE TIPOARR= ARRAY [1..2, 1..TAMEDADES ] OF REAL ;
  VAR
    LST: INTERACTIVE;
    NOMBRE: STRING;
    EDADES, SEGEDADES: TIPOARR;
    ED, POBTOTAL, EDADMUJERES : REAL ;
    EDAD, VAL : REAL ; (* AUXILIAR *)
    COL, REN, I, J, K : INTEGER; (* AUXILIARES *)
    ARRAUX: ARRAY [1..TAMEDADES] OF REAL;

  PROCEDURE ASIGNA( VAL : REAL );
  (* SIRVE PARA ASIGNAR VALORES AL ARREGLO EDADES. SI YA NO HAY LUGAR
     EN DONDE ASIGNAR VALORES, ENTONCES ESTOS VALORES NO SON TOMADOS
     EN CUENTA, TAL Y COMO SUCEDE EN EL LENGUAJE ESTADISTICO *)
   BEGIN
     IF COL <= TAMEDADES THEN BEGIN
       EDADES[REN, COL ]:=VAL;
       COL:=COL+1;
     END ELSE
       IF REN=1 THEN BEGIN
         REN:=2; COL:=2;
         EDADES[2,1]:=VAL;
       END;
   END;

 PROCEDURE ESCARR( EDADES : TIPOARR);
  BEGIN
    FOR J:=1 TO 2 DO BEGIN
      WRITELN(LST,' RENGLON: ',J);
      FOR K:=1 TO TAMEDADES DO
        WRITE(LST, EDADES[J, K]:10:2 );
      WRITELN(LST); WRITELN(LST);
    END;
  END;

 BEGIN
   WRITE('DISPOSITIVO DE SALIDA: ');
   READLN(NOMBRE);
   REWRITE(LST,NOMBRE);
   (* INICIALIZACION DE EDADES *)
  (* PRIMERO INICIALIZO AL ARREGLO COMO EN EL LENGUAJE ESTADISTICO *)
   FOR J:=1 TO 2 DO
     FOR K:=1 TO TAMEDADES DO
       EDADES[J,K]:=0;
   REN:=1; COL:=1;
   FOR J:=1 TO 3 DO BEGIN
     ASIGNA(1);
     ASIGNA(7);
     FOR K:=1 TO 3 DO BEGIN
       ASIGNA(3.14159); (* PI *)
       ASIGNA(2.71828); (* E *)
       ASIGNA(13);
     END;
     ASIGNA(11);
       ;
       N(LST,' EDADES');
```

```
(* SUMA DE LAS EDADES DE LAS MUJERES ENTRE 5 Y 90 AÑOS *)
EDADMUJERES:=0;
FOR I:=1 TO TAMEDADES DO BEGIN
  EDAD:=EDADES[1,I];
  IF (EDAD>=5) AND (EDAD<=90) THEN
    EDADMUJERES:=EDADMUJERES+EDAD;
END;
WRITELN(LST,'SUMA DE EDADES: ',EDADMUJERES);

(* INICIALIZACION DE SEGEDADES *)
(* PRIMERO INICIALIZO AL ARREGLO COMO EN EL LENGUAJE ESTADISTICO *)
FOR J:=1 TO 3 DO
  FOR I:=1 TO TAMEDADES DO
    SEGEDADES[J,I]:=0;

(* HAGO UNA ASIGNACION TEMPORAL A ARRAUX PORQUE EN EL PROGRAMA DEL
   LENGUAJE ESTADISTICO SE CREA UN ARREGLO EXPLICITO QUE NO ES
   NECESARIO *)
ARRAUX[1]:=EDADES[1,A];
ARRAUX[2]:=EDADES[1,[illegible]];
ARRAUX[3]:=EDADES[1,3];
ARRAUX[4]:=EDADES[1,4];
ARRAUX[5]:=EDADES[1,5];
ARRAUX[6]:=EDADES[1,A];
ARRAUX[7]:=EDADES[1,[illegible]];
ARRAUX[[illegible]]:=0;
SEGEDADES[2,A]:=ARRAUX[1];
SEGEDADES[2,[illegible]]:=ARRAUX[2];
SEGEDADES[2,A]:=ARRAUX[[illegible]];
SEGEDADES[2,[illegible]]:=ARRAUX[[illegible]];
SEGEDADES[2,8]:=ARRAUX[5];
SEGEDADES[[illegible],1]:=ARRAUX[6];
SEGEDADES[2,2]:=ARRAUX[7];
SEGEDADES[2,3]:=ARRAUX[[illegible]];
WRITELN(LST,'SEGEDADES');
ESCARR(SEGEDADES);

END.
```

Figura 6.21 Reseña Semántica del Programa de la Figura 6.20

```
PONNIV :   APVDT 45 APDIS 1 INFO[APDIS] 1
DESCRI  : APVDT 45 APDES 0 CP 1 CS 3 CT 270 CC 0
VE DESCRIPTOR    : DIRVDT 45 LIGA 1 INFO 1
 DESCRIPTOR DE  45 :  CP 1 CS 3 CT 270 CC 0
PONNIV :   APVDT 47 APDIS 2 INFO[APDIS] 6
DESCRI  : APVDT 47 APDES 5 CP 1 CS 17 CT 1 CC 2
DESCRI  : APVDT 0 APDES 10 CP 1 CS 2 CT 0 CC 0
DESCRI  : APVDT 0 APDES 15 CP 1 CS 8 CT 0 CC 0
PONNIV :   APVDT 48 APDIS 3 INFO[APDIS] 21
DESCRI  : APVDT 48 APDES 20 CP 1 CS 17 CT 20 CC 2
DESCRI  : APVDT 0 APDES 25 CP 1 CS 2 CT 0 CC 0
DESCRI  : APVDT 0 APDES 30 CP 1 CS 8 CT 0 CC 0
PONNIV :   APVDT 51 APDIS 4 INFO[APDIS] 36
DESCRI  : APVDT 51 APDES 35 CP 1 CS 4 CT 39 CC 0
PONNIV :   APVDT 52 APDIS 5 INFO[APDIS] 41
DESCRI  : APVDT 52 APDES 40 CP 1 CS 4 CT 40 CC 0
PONNIV :   APVDT 53 APDIS 6 INFO[APDIS] 46
DESCRI  : APVDT 53 APDES 45 CP 1 CS 4 CT 41 CC 0
INICIO DE ASIGNACION. ANALISIS DEL LADO DERECHO
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
INICIO DE ARREGLO EXPLICITO
VE DESCRIPTOR    : DIRVDT 45 LIGA 1 INFO 1
 DESCRIPTOR DE  45 :  CP 1 CS 3 CT 270 CC 0
METE OPERADOR :   IOPDOR 1 OPERACION MASUN
METE OPERANDO :   IOPNDO 0 TIPO 1 LOC.   1.00000
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 1 OPERACION MASUN
SACA OPERANDO :   IOPNDO 0 TIPO 1 LOC.   1.00000
METE OPERANDO :   IOPNDO 0 TIPO 4 LOC. -1 NIV 7885
METE OPERADOR :   IOPDOR 1 OPERACION MASUN
METE OPERANDO :   IOPNDO 1 TIPO 1 LOC.   7.00000
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 1 OPERACION MASUN
SACA OPERANDO :   IOPNDO 1 TIPO 1 LOC.   7.00000
METE OPERADOR :   IOPDOR 1 OPERACION MASUN
METE OPERANDO :   IOPNDO 1 TIPO 1 LOC.   3.14159
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 1 OPERACION MASUN
SACA OPERANDO :   IOPNDO 1 TIPO 1 LOC.   3.14159
METE OPERADOR :   IOPDOR 1 OPERACION MASUN
METE OPERANDO :   IOPNDO 1 TIPO 1 LOC.   2.71828
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 1 OPERACION MASUN
SACA OPERANDO :   IOPNDO 1 TIPO 1 LOC.   2.71828
METE OPERADOR :   IOPDOR 1 OPERACION MASUN
METE OPERANDO :   IOPNDO 1 TIPO 1 LOC.   1.30000E1
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 1 OPERACION MASUN
SACA OPERANDO :   IOPNDO 1 TIPO 1 LOC.   1.30000E1
METE OPERADOR :   IOPDOR 1 OPERACION MASUN
METE OPERANDO :   IOPNDO 1 TIPO 1 LOC.   1.10000E1
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 1 OPERACION MASUN
SACA OPERANDO :   IOPNDO 1 TIPO 1 LOC.   1.10000E1
FIN DE ARREGLO EXPLICITO
10 ELEMENTOS GRABADOS A ARREXP
NO. DE DIMENSIONES: 2
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
```

DECLARACION DE VARIABLES

Inicia proposición de asignación.

Análisis de un Arreglo Explícito. Llamado (5036).

Constante TAMEDADES

Se toma expresión: 1

Operando Temporal. Corresponde al Arreglo Explícito.

Expresión: 7

Expresión: PI

Termina el análisis del Arreglo Explícito. Contiene 2 dimensiones. Se generó al código (ARREXP) 10 elementos.

```
 DESCRIPTOR DE  47 :  CP 1 CS 17 CT 1 CC 2
METE OPERANDO :   IOPNDO 1 TIPO 4 LOC. 1 NIV 1
SACA OPERANDO :   IOPNDO 1 TIPO 4 LOC. 1 NIV 1
SACA OPERANDO :   IOPNDO 0 TIPO 4 LOC. -1 NIV 7885
FIN DE ASIGNACION
TOPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
INICIO DE ASIGNACION. ANALISIS DEL LADO DERECHO
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
METE OPERADOR :   IOPDOR 1 OPERACION MASUN
METE OPERADOR :   IOPDOR 2 OPERACION MASUN
METE OPERADOR :   IOPDOR 3 OPERACION MASUN
VE DESCRIPTOR     : DIRVDT 47 LIGA 2 INFO 6
 DESCRIPTOR DE  47 :  CP 1 CS 17 CT 1 CC 2
METE OPERANDO :   IOPNDO 0 TIPO 4 LOC. 1 NIV 1
INICIO DE EXP. SECTIVA DE ARREGLO
SACA OPERANDO :   IOPNDO 0 TIPO 4 LOC. 1 NIV 1
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 4 OPERACION MASUN
METE OPERANDO :   IOPNDO 0 TIPO 1 LOC.  1.00000
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 4 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
SACA OPERANDO :   IOPNDO 0 TIPO 1 LOC.  1.00000
METE OPERANDO :   IOPNDO 0 TIPO 4 LOC. -1 NIV 7885
FIN DE EXP. SELECTIVA DE ARREGLO
5 ELEMENTOS GRABADOS A SELEC
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 3 OPERACION MASUN
METE OPERADOR :   IOPDOR 3 OPERACION MAYOIG
METE OPERADOR :   IOPDOR 4 OPERACION MASUN
METE OPERANDO :   IOPNDO 1 TIPO 1 LOC.  5.00000
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 4 OPERACION MASUN
OPERACION BINARIA
SACA OPERADOR :   IOPDOR 3 OPERACION MAYOIG
SACA OPERANDO :   IOPNDO 1 TIPO 1 LOC.  5.00000
SACA OPERANDO :   IOPNDO 0 TIPO 4 LOC. -1 NIV 7885
METE OPERANDO :   IOPNDO 0 TIPO 4 LOC. -1 NIV 7885
METE OPERADOR :   IOPDOR 3 OPERACION CONJUNC
METE OPERADOR :   IOPDOR 4 OPERACION MASUN
VE DESCRIPTOR     : DIRVDT 47 LIGA 2 INFO 6
 DESCRIPTOR DE  47 :  CP 1 CS 17 CT 1 CC 2
METE OPERANDO :   IOPNDO 1 TIPO 4 LOC. 1 NIV 1
INICIO DE EXP. SECTIVA DE ARREGLO
SACA OPERANDO :   IOPNDO 1 TIPO 4 LOC. 1 NIV 1
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 5 OPERACION MASUN
METE OPERANDO :   IOPNDO 1 TIPO 1 LOC.  1.00000
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 5 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
SACA OPERANDO :   IOPNDO 1 TIPO 1 LOC.  1.00000
METE OPERANDO :   IOPNDO 1 TIPO 4 LOC. -1 NIV 7885
FIN DE EXP. SELECTIVA DE ARREGLO
5 ELEMENTOS GRABADOS A SELEC
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 4 OPERACION MASUN
METE OPERADOR :   IOPDOR 4 OPERACION MENOIG
METE OPERADOR :   IOPDOR 5 OPERACION MASUN
METE OPERANDO :   IOPNDO 2 TIPO 1 LOC.  9.00000E1
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 5 OPERACION MASUN
OPERACION BINARIA
```

Handwritten annotations:

- ARREGLO EDADES
- ARREGLO EXPLICITO A ASIGNAR.
- INICIA ASIGNACIÓN A EDADMUJERES.
- EDADES
- EDADES[1,] A SELEC SE GENERARON 5 ELEMENTOS.
- 5 (Expresión)
- EDADES[1,]>=5 (LLAMADO 5025)
- EDADES[1,] En SELEC se generaron 5 elementos.

```
SACA OPERADOR :   IOPDOR 4 OPERACION MENOIG
SACA OPERANDO :   IOPNDO 2 TIPO 1 LOC.  9.00000E1
SACA OPERANDO :   IOPNDO 1 TIPO 4 LOC. -1 NIV 7885
METE OPERANDO :   IOPNDO 1 TIPO 4 LOC. -1 NIV 7885
OPERACION BINARIA
SACA OPERADOR :   IOPDOR 3 OPERACION CONJUNC
SACA OPERANDO :   IOPNDO 1 TIPO 4 LOC. -1 NIV 7885
SACA OPERANDO :   IOPNDO 0 TIPO 4 LOC. -1 NIV 7885
METE OPERANDO :   IOPNDO 0 TIPO 4 LOC. -1 NIV 7885
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 2 OPERACION MASUN
METE OPERADOR :   IOPDOR 2 OPERACION MULT
VE DESCRIPTOR   : DIRVDT 47 LIGA 2 INFO 6
 DESCRIPTOR DE  47 :  CP 1 CS 17 CT 1 CC 2
METE OPERANDO :   IOPNDO 1 TIPO 4 LOC. 1 NIV 1
INICIO DE EXP. SECTIVA DE ARREGLO
SACA OPERANDO :   IOPNDO 1 TIPO 4 LOC. 1 NIV 1
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 3 OPERACION MASUN
METE OPERANDO :   IOPNDO 1 TIPO 1 LOC.  1.00000
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 3 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
SACA OPERANDO :   IOPNDO 1 TIPO 1 LOC.  1.00000
METE OPERANDO :   IOPNDO 1 TIPO 4 LOC. -1 NIV 7885
FIN DE EXP. SELECTIVA DE ARREGLO
5 ELEMENTOS GRABADOS A SELEC
OPERACION BINARIA
SACA OPERADOR :   IOPDOR 2 OPERACION MULT
SACA OPERANDO :   IOPNDO 1 TIPO 4 LOC. -1 NIV 7885
SACA OPERANDO :   IOPNDO 0 TIPO 4 LOC. -1 NIV 7885
METE OPERANDO :   IOPNDO 0 TIPO 4 LOC. -1 NIV 7885
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 1 OPERACION MASUN
INICIO DE EXP. SECTIVA DE ARREGLO
SACA OPERANDO :   IOPNDO 0 TIPO 4 LOC. -1 NIV 7885
METE OPERANDO :   IOPNDO 0 TIPO 4 LOC. -1 NIV 7885
FIN DE EXP. SELECTIVA DE ARREGLO
2 ELEMENTOS GRABADOS A SELEC
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
ANALISIS DE LA PARTE IZQUIERDA DE LA ASIGNACION
VE DESCRIPTOR   : DIRVDT 53 LIGA 6 INFO 46
 DESCRIPTOR DE  53 :  CP 1 CS 4 CT 41 CC 0
METE OPERANDO :   IOPNDO 1 TIPO 2 LOC. 41 NIV 1
SACA OPERANDO :   IOPNDO 1 TIPO 2 LOC. 41 NIV 1
SACA OPERANDO :   IOPNDO 0 TIPO 4 LOC. -1 NIV 7885
FIN DE ASIGNACION
TOPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
INICIO DE ASIGNACION. ANALISIS DEL LADO DERECHO
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
INICIO DE ARREGLO EXPLICITO
VE DESCRIPTOR   : DIRVDT 45 LIGA 1 INFO 1
 DESCRIPTOR DE  45 :  CP 1 CS 3 CT 270 CC 0
METE OPERADOR :   IOPDOR 1 OPERACION MASUN
VE DESCRIPTOR   : DIRVDT 47 LIGA 2 INFO 6
 DESCRIPTOR DE  47 :  CP 1 CS 17 CT 1 CC 2
METE OPERANDO :   IOPNDO 0 TIPO 4 LOC. 1 NIV 1
INICIO DE EXP. SECTIVA DE ARREGLO
SACA OPERANDO :   IOPNDO 0 TIPO 4 LOC. 1 NIV 1
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 2 OPERACION MASUN
METE OPERANDO :   IOPNDO 0 TIPO 1 LOC.  1.00000
OPERACION UNARIA
```

Handwritten annotations (right margin):

- EDADES[1,]<=90
- (...) Y (...)
- EDADES[1,]
- (...)*EDADES[1,]
- (...)[.]
- EDADMUJERES
- Temporal. Expresión a Asignar.
- Asignación a SEGEDADES

Figura 6.22 Código Correspondiente al Programa de la Figura 6.20

```
(*$S++*)
(*$G+*)
PROGRAM LENGEST;
USES TRANSCEN, CODGRAL, INICOD;      <---- Rutinas de Biblioteca.
LABEL 1, 2, 3, 4, 5, 6, 7, 8, 9, 10;
PROCEDURE INI11;
BEGIN
INICIAL1;                            <---- Inicializa memoria. Lee SEMCHARS, etc.
NIVEL:=1;
DEC1ARR(1,2,2);      <--+
DEC2ARR(8);             |
DEC1ARR(20,2,2);        |
DEC2ARR(8);          <--+------------ Asigna Descripción Dinámica a Arreglos.
TOPEMEM:=DISPLAY[1]+42;  <----------- Tope al stack de Memoria.
END; (* INI11 *)
BEGIN
INI11;
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
ARREXP[1]:=3;                  --+
AXARREXP( 1.00000,2);            |
BXARREXP(4,2,19,2);              |
DXARREXP(8);                     |
AXARREXP( 7.00000,3);            |
ARREXP[4]:=3;                    |  Construcción del Arreglo Explícito.
AXARREXP( 3.14159,5);            }  En CONSARREXP!
AXARREXP( 2.71828,6);            |     No. de elementos en ARREXP = 10.
AXARREXP( 1.30000E1,7);          |
ARREXP[8]:=10000;                |
AXARREXP( 1.10000E1,9);          |
ARREXP[10]:=10000;               |
CONSARREXP(10);                --+
SELEC[0]:=0;                   --}  Asignación sin Selección.
MUEVE(1,4,1,-1,4);             --+
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
SELEC[2]:=1;                   --+
SELEC[3]:=-1000;                 |
SELEC[1]:=1;                     |
SELEC[4]:=1;                     }  EDADES[1, ]
SELEC[5]:=-1002;                 |  En CONSELARR: NIVEL= NIVELUNO, TIPO = 4
SELEC[0]:=2;                     |     LOCALIDAD = 1.
NIVELUNO:=1;                     |
CONSELARR(1,4);                --+
OPERBIN(MAYOIG,-1,4, 5.00000,1);  <---- EDADES[1, ] >= 5
SELEC[2]:=1;
SELEC[3]:=-1000;
SELEC[1]:=1;
SELEC[4]:=1;
SELEC[5]:=-1002;
SELEC[0]:=2;
NIVELUNO:=1;
CONSELARR(1,4);
OPERBIN(MENOIG,-1,4, 9.00000E1,1);
OPERBIN(CONJUNC,-1,4,-1,4);  <-------- (Temporal) Y (Temporal)
SELEC[2]:=1;
SELEC[3]:=-1000;
SELEC[1]:=1;
SELEC[4]:=1;
SELEC[5]:=-1002;
SELEC[0]:=2;
```

```
CONSELARR(1,4);
OPERBIN(MULT,-1,4,-1,4);
```

← (Temporal) * Temporal

```
SELEC[1]:=1;
SELEC[2]:=-1001;
SELEC[0]:=1;
CONSELARR(-1,4);
```

} (Temporal) [.]

```
SELEC[0]:=0;
MUEVE(41,2,1,-1,4);
```

] Asignación.

```
TOPEMEM:=AUXTOP1;  (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM;  (* INICIA EXPRESION O ASIGNACION *)
SELEC[2]:=1;
SELEC[3]:=-1000;
SELEC[1]:=1;
SELEC[5]:=6;
SELEC[6]:=-1000;
SELEC[7]:=2;
SELEC[8]:=6;
SELEC[9]:=1;
SELEC[10]:=-1000;
SELEC[4]:=3;
SELEC[0]:=2;
NIVELUNO:=1;
CONSEL ARR(1,4);
```

} Selección de EDADES

```
ARREXP[1]:=-ISTOPNDO;
BXARREXP(4,1,10,8);
CONSARREXP(1);
```

} Construcción de Arreglo Explícito.

```
SELEC[2]:=2;
SELEC[3]:=-1000;
SELEC[1]:=1;
SELEC[5]:=4;
SELEC[6]:=8;
SELEC[7]:=1;
SELEC[8]:=3;
SELEC[4]:=2;
SELEC[0]:=2;
MUEVE(20,4,1,-1,4);
```

} Asignación a elementos seleccionados de SEGEDADES.

```
TOPEMEM:=AUXTOP1;  (* FIN DE ASIGNACION *)
CLOSE(LST,LOCK);
END.
```

Figura 6.23 Ejecución del Programa Objeto de la Figura 6.22

```
METE DES. IND 0 TIPO 2 LOC. 42
METE DES. IND 1 TIPO 4 LOC. 43
METE DES. IND 2 TIPO 2 LOC. 62      } Stack de Operandos (STOPNDO)
METE DES. IND 3 TIPO 2 LOC. 63
METE DES. IND 4 TIPO 2 LOC. 64
METE DES. IND 5 TIPO 2 LOC. 65
METE DES. IND 6 TIPO 2 LOC. 66
APAE FINAL 11 <------ (No. de Elementos + 1) en ARREXP
ARREXP : I 0     10 *
I 1     3 * I 2     -1 * I 3     -3 * I 4      3 * I 5     -4 *   ] Contenido de
I 6    -5 * I 7     -6 * I 8  10000 * I 9     -7 * I 10  10000 *  ] ARREXP

TOPEMEM= 62 (Tope al Stack de Memoria -MEM-)
RESOPNDO -1.00000 4 SACA DES. IND 0 TIPO 4 LOC. 43 (loc = -1.0 (TEMPORAL), tipo=4).
43 4 0          SE SACA UN ELEMENTO DE STOPNDO (LOC= 43, TIPO=4, El último
RESOPNDO  1.00000 4 1 4 0               cero es porque no es escalar).
********* MUEVE *** OPNDO1 4 1 OPNDO2 4 43
                     └── (TIPO=4, LOC=1) <── (TIPO=4, LOC=43)
   ESCARREGLO (Contenido de EDADES).
LOC 1 NO. DIM. 2
DIMENSIONES :   2  8 <── Tamaño de cada dimensión.

 SECCION :   1 (Renglón 1)

     1.00      7.00      3.14      2.72     13.00      3.14      2.72     13.00

 SECCION :   2 (Renglón 2)

     3.14      2.72     13.00     11.00      1.00      7.00      3.14      2.72
RESOPNDO  1.00000 4 1 4 0 (lc=1.0, tp=4. LOC= 1.0, TIPO=4, NO ES ESCALAR (0))

SELEC : -1: 0 0: 2 - Contenido de SELEC y SELECB
#1     1     1 #2     1     1 #3 -1000     1 #4     1     1 #5 -1002     -8
(Indice, elemento de SELEC, elemento de SELECB).
LOCSEL : #1  1 #2  4 ] Contenido de LOCSEL y LOCRES
LOCRES : #1  2       ]
CONS EL ARR.  FUENTE : 1 4 DESTINO : 42 4  ] Resultado de la Selección.
METE DES. IND 0 TIPO 4 LOC. 42             ] TIPO= 4, LOC= 42
LONG.:  TIPO 4 LOC. 42 LONGITUD 10         ] longitud Total = 10 celdas.
RESOPNDO  5.00000 1 0 1 1 (lc = 5.0, tp= 1, LOC = 0 (LITERAL), TIPO= 1, ESCALAR (1))
COMOPNDO  5.00000 (VALOR DEL ESCALAR = 5.0)
RESOPNDO -1.00000 4 SACA DES. IND 0 TIPO 4 LOC. 42
42 4 0 TEMPORAL. TIPO = 4, LOC = 42, NO ES ESCALAR (0).
OP. BIN. 4 OPNDO1 4 42  1.30000E1 OPNDO2 1 0  5.00000 RES 4 52  1.00000 (Op. Bina
METE DES. IND 0 TIPO 4 LOC. 52            ria no. 4. Operando 1: TIPO= 4, LOC=42
LONG.:  TIPO 4 LOC. 52 LONGITUD 10        Operando 2: TIPO= 1, LOC= 0, VALOR
RESOPNDO  1.00000 4 1 4 0                  DEL ESCALAR = 5.0. Resultado: TIPO=4
                                           LOC= 52.
SELEC : -1: 0 0: 2
#1     1     1 #2     1     1 #3 -1000     1 #4     1     1 #5 -1002     -8

LOCSEL : #1  1 #2  4
LOCRES : #1  2
CONS EL ARR.  FUENTE : 1 4 DESTINO : 62 4
METE DES. IND 1 TIPO 4 LOC. 62
LONG.:  TIPO 4 LOC. 62 LONGITUD 10
RESOPNDO  9.00000E1 1 0 1 1 (lc= 90, tp= 1 (LITERAL), LOC=0, TIPO=1, ESCALAR (1))
COMOPNDO  9.00000E1 VALOR DEL ESCALAR = 90.0
RESOPNDO -1.00000 4 SACA DES. IND 1 TIPO 4 LOC. 62
62 4 0
OP. BIN. 0 OPNDO1 4 62  1.30000E1 OPNDO2 1 0  9.00000E1 RES 4 72  1.00000
```

```
METE DES. IND 1 TIPO 4 LOC. 72
LONG.:  TIPO 4 LOC. 72 LONGITUD 10
RESOPNDO -1.00000 4 SACA DES. IND 1 TIPO 4 LOC. 72
72 4 0
RESOPNDO -1.00000 4 SACA DES. IND 0 TIPO 4 LOC. 52
52 4 0
OP. BIN. 12 OPNDO1 4 52  1.00000 OPNDO2 4 72  1.00000 RES 4 82  1.00000
METE DES. IND 0 TIPO 4 LOC. 82
LONG.:  TIPO 4 LOC. 82 LONGITUD 10
RESOPNDO  1.00000 4 1 4 0

SELEC : -1: 0 0: 2
#1      1      1 #2      1      1 #3 -1000      1 #4      1      1 #5 -1002     -8

LOCSEL : #1  1 #2  4
LOCRES : #1  2
CONS EL ARR.  FUENTE : 1 4 DESTINO : 92 4
METE DES. IND 1 TIPO 4 LOC. 92
LONG.:  TIPO 4 LOC. 92 LONGITUD 10
RESOPNDO -1.00000 4 SACA DES. IND 1 TIPO 4 LOC. 92
92 4 0
RESOPNDO -1.00000 4 SACA DES. IND 0 TIPO 4 LOC. 82
82 4 0
OP. BIN. 9 OPNDO1 4 82  1.00000 OPNDO2 4 92  1.30000E1 RES 4 102  1.30000E1
METE DES. IND 0 TIPO 4 LOC. 102
LONG.:  TIPO 4 LOC. 102 LONGITUD 10
RESOPNDO -1.00000 4 SACA DES. IND 0 TIPO 4 LOC. 102
102 4 0

SELEC : -1: 0 0: 1
#1      1      1 #2 -1001     -8
LOCSEL : #1  1
LOCRES :
CONS EL ARR.  FUENTE : 102 4 DESTINO : 112 2
METE DES. IND 0 TIPO 2 LOC. 112
LONG.:  TIPO 2 LOC. 112 LONGITUD 1
SELEC. DE ARR.  VALOR EN DESTINO :  3.30000E1
RESOPNDO -1.00000 4 SACA DES. IND 0 TIPO 2 LOC. 112 <—— sale elemento de
112 2 1                                 STOPNDO: (Tipo = 2 (escalar real), Loc = 112).
COMOPNDO  3.30000E1 <———————————————————— VALOR DEL ESCALAR.
RESOPNDO  4.10000E1 2 41 2 1
********* MUEVE *** OPNDO1 2 41 OPNDO2 2 112VALOR  3.30000E1 : (LOC = 41, TIPO =
RESOPNDO  1.00000 4 1 4 0                   2 (ESCALAR REAL)) <—— (LOC = 112, TIPO = 2).
                                            RESULTADO = 33.0
SELEC : -1: 0 0: 2
#1      1      1 #2      1      1 #3 -1000      1 #4      3      3 #5      6      6
#6 -1000      6 #7      2      2 #8      6      6 #9      1      1 #10 -1000      1

LOCSEL : #1  1 #2  4
LOCRES : #1  2
CONS EL ARR.  FUENTE : 1 4 DESTINO : 42 4
METE DES. IND 0 TIPO 4 LOC. 42
LONG.:  TIPO 4 LOC. 42 LONGITUD 9
METE DES. IND 1 TIPO 4 LOC. 51
APAE FINAL 2
ARREXP : I 0       1 *
I 1     -1 *
TOPEMEM= 61
RESOPNDO  2.00000E1 4 20 4 0
********* MUEVE *** OPNDO1 4 20 <—— Asignación con Selección.
SELEC : -1: 0 0: 2
#1      1      1 #2      2      2 #3 -1000      2 #4      2      2 #5      4      4
#6      8      8 #7      1      1 #8      3      3
LOCSEL : #1  1 #2  4
LOCRES : #1  2
RESOPNDO -1.00000 4 SACA DES. IND 0 TIPO 4 LOC. 51
```

```
51 4 0
 OPNDO2 4 51
****** ERROR *****  6
ASIG-SELEC-ARR. OPNDO1: 20 4  OPNDO2: 51 4
   ESCARREGLO
LOC 20 NO. DIM. 2
DIMENSIONES :    2  8

 SECCION :    1

     0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00

 SECCION :    2

     3.14      1.00      0.00      3.14      7.00      3.14      2.72     13.00
```

Handwritten annotations: ← porque (Arreglo Renglón ← Arreglo Columna) ((LOC=20, TIPO=4) ← (LOC=51, TIPO=4)) Contenido de SEGEDADES, después de la asignación.

Figura 6.23.A Ejecución del Programa en PASCAL de la figura 6.20.A .

```
 EDADES
 RENGLON: 1
     1.00      7.00      3.14      2.72     13.00      3.14      2.72     13.00

 RENGLON: 2
     3.14      2.72     13.00     11.00      1.00      7.00      3.14      2.72

 SUMA DE EDADES:  3.30000E1
 SEGEDADES
 RENGLON: 1
     0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00

 RENGLON: 2
     3.14      1.00      0.00      3.14      7.00      3.14      2.72     13.00
```

F) PROPOSICION DE ASIGNACION

De entre las distintas proposiciones que comprende el lenguaje estadístico, la proposición de asignación es la más importante, pues casi siempre constituye la acción que sigue a la evaluación de una expresión y además permite la transferencia de datos entre las distintas variables que viven en el programa fuente.

La definición gramatical de la asignación se exhibe a continuación:

<Asig> :: = <5035> <Est Ref> ":=" <5033>
<Exp> <5034>
<Est Ref> :: = <Identif> <Oper Estruct> <Exp Selec>
<Oper Estruct> :: = "." <Op Est> |e
<Op Est> :: = "TITREN" | "TITCOL" | "NOMREN" |
"NOMCOL" | "MARCOREN" |
"MARCOCOL" | "TITULO" |
"CUERPO"
<Exp Selec> :: = <Exp Selec de Arr> |e

Del lado izquierdo de la asignación puede haber un identificador aislado que corresponda a un escalar, a un arreglo o a una estructura, también puede estar un arreglo acompañado de una expresión selectiva y también se puede hacer referencia a uno de los campos de una estructura, tabla, histograma o gráfica.

En la otra parte de la asignación, en el lado derecho, debe ir una expresión cuya estructura resultante sea del mismo tipo y forma que la estructura en el lado izquierdo y aún más, si las estructuras corresponden a arreglos, ambas deben coincidir en el número de dimensiones.

La gramática original sufrió varios cambios, entre los que se encuentran las adiciones de los llamados semánticos.

En primer lugar, la definición de la estructura de referencia (Est Ref) fué modificada. La definición original se encontraba así:

<Est Ref> :: = <Identif> <Oper Estruct>
<Oper Estruct> :: = "." <Op Est> | <Exp Selec>

Como puede verse, no se podía referenciar a un campo de estructura que fuera un arreglo, con una expresión selectiva de arreglo, como en el caso de TAB.NOMREN[1], donde TAB es una tabla.

Ya en la gramática actual, si se puede tener esta clase de referencias.

Los otros cambios residen en la inclusión de los llamados semánticos 5033, 5034 y 5035.

Al llamado 5033 le corresponden dos tareas a realizar. Inicializa a los topes de los stack de operandos y operadores (IOPNDO y IOPDOR respectivamente), a fin de eliminar la posible información que haya quedado, en caso de haber fallado el análisis de la expresión anterior.

Además, este llamado se encarga de generar la asignación "AUXTOP1 : = TOPEMENM;" donde AUXTOP1 es una variable auxiliar en el programa objeto que se emplea para recuperar todo el espacio en memoria, que ocupan los temporales generados durante la evaluación de una expresión.

En el caso de la proposición de asignación, al finalizar la realización de ésta, se reestablecerá el tope al stack de memoria (TOPEMEM) mediante esta variable auxiliar.

Cabe aclarar que AUXTOP1 es una variable global al programa objeto y que funcionará sin problemas mientras no se implanten llamados a funciones.

Si dentro de una expresión se hace un llamado a una función y luego ésta se llama así misma, se requiere de tantas variables AUXTOP1 como llamados ocurran, de tal manera que una posible alternativa para manejar a AUXTOP1 sería que al llamar a una función, el valor de AUXTOP1 se almacenara dentro de los datos de la rutina. Al terminar la ejecución de la rutina se reestablecería el valor a AUXTOP1. De esta forma nunca se perdería el contenido de la variable auxiliar al evaluar expresiones.

En otros contextos del lenguaje, por ejemplo en una proposición condicional, que contiene una expresión, también se hace uso de

la variable AUXTOP1 y durante la compilación se hace un llamado 5033 antes de procesar a la expresión de la condición.

El llamado 5034, como ya se comentó en la sección dedicada a expresiones, se emplea para indicar al analizador semántico que finaliza una expresión.

Finalmente el tercer llamado y última modificación, lo constituye el llamado 5035 que se encarga de procesar a toda la proposición de asignación.

Dentro de la evaluación de la proposición, primeramente se manda procesar a la expresión y dentro de ese análisis se ejecuta el llamado 5033 y todos los demás involucrados en la evaluación de la expresión. Al regresar de la rutina que procesa a la expresión, ya se puede generar el código para llevar a cabo la asignación y también se hace una validación parcial, referente a que las dos estructuras que participan en la asignación, sean ambas reales o textos.

Hay dos rutinas que se generan para efectuar una asignación en el programa objeto. La primera rutina, que es la más utilizada, se llama MUEVE y el encabezado de su declaración se presenta a continuación:

```
PROCEDURE MUEVE (LC1, TP1, NIV1 : INTEGER;
                 LC2: REAL; TP2: INTEGER);
```

Como el operando que recibe la asignación debe ser forzosamente una variable, entonces se dispone de los tres primeros parámetros para describirla. TP1 indica el tipo de la estructura. La dirección en memoria se entrega en NIV1 y LC1 que contienen el nivel léxico y el desplazamiento dentro de ese nivel, respectivamente. En lo que respecta a la expresión a asignar, ésta puede ser una literal numérica, una variable o un temporal. TP2 indica si se tiene una literal numérica (TP2=1) o una variable, mientras que LC2 contiene el valor de la literal (por eso es real), un -1, para indicar que es un temporal (cuya verdadera representación deberá extraerse del stack de operandos STOPNDO) o la localidad de la variable con respecto al nivel en

donde vive, donde el nivel se maneja en la variable global (NIVDOS), tal y como ocurre en las operaciones binarias y unarias (OPERBIN y OPERUNA).

La otra rutina se llama ASIGEST y se emplea para asignar una estructura gráfica, tabla o histograma a otra igual. Cabe indicar que estos tipos de estructuras no pueden participar en operación alguna, ni se les puede aplicar una expresión selectiva, puesto que son variables compuestas.

La rutina ASIGEST hace llamados a MUEVE para ir asignando cada uno de los campos en las estructuras, de esta forma se genera menos código que si se escribieran varias lineas de "MUEVE".

El encabezado de la declaración de ASIGEST se presenta a continuación:

```
PROCEDURE ASIGEST (TP, LC1, NV1, LC2, NV2:
                              INTEGER);
```

En TP se indica el tipo de la estructura y como los dos operandos en la asignación corresponden a variables, LC1 y NV1 comprenden la dirección en memoria del operando izquierdo, como sucede en MUEVE y por otra parte LC2 y NV2 contienen la dirección del operando a asignar.

Cuando el operando izquierdo está afectado por una expresión selectiva de arreglo, la rutina MUEVE llama a los procedimientos DECODSELEC y RUTSEL para realizar la asignación, tal y como se expuso en la sección anterior, dedicada a expresiones. Si no hay una expresión selectiva y las estructuras en la asignación son arreglos, entonces se llama a la rutina MUEVEARR.

Para saber cuando hay una expresión selectiva de arreglo, la rutina MUEVE se fija en el elemento cero del arreglo SELEC, el cual contiene el número de dimensiones que se representan en el mismo. Si dicho valor es cero, entonces significa que no hay selección de elementos. Así pues, cuando no hay una expresión selectiva, se genera la asignación: "SELEC[0]: = 0;" antes de generar el llamado a MUEVE.

En caso de que en una o varias dimensiones, sobren elementos a

asignar, éstos ya no se toman en cuenta y sí por el contrario, queden elementos que no tengan que se les asigne, éstos se dejan inalterados.

Después de efectuar la asignación, se reestablece el tope al stack de memoria (TOPEMEM), mediante la variable auxiliar AUXTOP1.

Si se desea ver un ejemplo del uso de la proposición de asignación, de la reseña semántica de su análisis y del código generado, ver el programa mostrado al final de la sección anterior.

G) PROPOSICION CONDICIONAL

La proposición condicional se implantó con la misma sintaxis y las mismas características semánticas con que originalmente se había diseñado. (Revisar el proyecto original en García G1). Se puede decir que es semejante a la proposición condicional de APL o de C, en donde la expresión condicional es de tipo aritmético, pero se interpreta como si fuera una expresión lógica o booleana.

En cuanto a la definición gramatical, sólo se agregaron algunos llamados semánticos. Veamos dichas modificaciones en la figura siguiente:

<Cond> : : = <P Si> <RC1>
<P Si> : : = "SI" <5033> <Exp> <5034> "ENTONCES'
<5044> <P Inc>
<RC1> : : = "SINO" <5046> <P Gral< | e

La proposición que se ejecuta en caso de que la expresión condicional resulte verdadera y que está representada en la gramática por el símbolo <P Inc> debe ser una proposición incondicional. Esto quiere decir que no se puede escribir una proposición de la forma: SI <Exp> ENTONCES SI <Exp> ENTONCES <Proposición> SINO <Proposición>. De esta manera se evita el clásico problema de ambigüedad que ocurre en la definición de la proposición condicional. Esa misma proposición debe escribirse: SI <Exp> ENTONCES INICIO SI <Exp> ENTONCES... .

Por otra parte, la proposición alternativa, que se ejecuta cuando la expresión condicional es falsa y que está representada en la gramática por el símbolo <P Gral>, puede ser cualquier proposición, incluyendo por supuesto a la proposición condicional. Por ejemplo, es totalmente legal escribir proposiciones del tipo: SI <Exp> ENTONCES <Proposición Incondicional> SINO SI <Exp> ENTONCES <Proposición Incondicional> SINO <Proposición>.

Hay más llamados semánticos que se agregaron en otros lugares de la gramática, a fin de poder analizar correctamente a la proposición condicional y son los siguientes:

```
<P> :: = <Decl>
         <L Etiq> <P Gral> <5047> |
          e
<TERMINA> :: = "FIN" <5045><RFI> <TERMINA> |
                 e
```

En el transcurso de la sección se explicará cual es el papel que cumplen cada uno de estos llamados semánticos.

Las proposiciones condicionales se traducen al programa objeto en otras proposiciones condicionales. Si en el programa fuente se tiene una proposición sencilla (SI <Exp> ENTONCES <P Inc>) o compuesta (SI <Exp> ENTONCES INICIO... FIN) sin alternativa (la parte correspondiente al SINO...), al programa objeto se generará:

```
IF CONDRES THEN BEGIN
:
END;
```

Y si lo que se tiene es una proposición sencilla o compuesta con alternativa, entonces se generará:

```
IF CONDRES THEN BEGIN
:
END ELSE BEGIN
:
END;
```

CONDRES es una variable lógica y global, que contiene el resultado de la interpretación lógica de la expresión de la condicional.

A fin de poder determinar cual es el papel que cumplen las proposiciones de "FIN" en el programa fuente se insertó el llamado 5045.

El llamado 5047 se introdujo por la necesidad de generar el "END" que cierre la traducción de una proposición condicional sencilla, ya que debe generarse hasta terminar de analizar el árbol sintáctico como en el caso de: SI A ENTONCES A: = NO A;

Para poder ir generando las líneas que componen a una proposición y detectar errores, se diseño un stack o pila de proposiciones que actualmente sólo se utiliza para administrar las proposiciones condicionales, pero en el futuro servirá para procesar a todas las demás proposiciones en el lenguaje.

El stack de proposiciones tiene la siguiente forma:

```
VAR STPROP: ARRAY [1.. TAMSTPROP]
            OF INTEGER;
```

TAMSTPROP es una constante.

Los elementos que maneja hasta el momento la pila de proposiciones son los siguientes:

0.- No hay proposición
1.- Condicional sencilla (SI <Exp> ENTONCES...)
2.- Condicional compuesta (SI <Exp> ENTONCES INICIO...)
3.- Fin de la primera parte de una condicional (Si <Exp> ENTONCES INICIO...FIN...)
4.- Alternativa sencilla de condicional (SINO...)
5.- Alternativa compuesta de condicional (SINO INICIO...)

El stack de proposiciones se inicializa en ceros cada vez que comienza un bloque o nivel lexicográfico y al finalizar éste se verifica que no haya proposiciones que queden sin cerrar, en cuyo caso se señala el error correspondiente.

Veamos como se emplea el stack de proposiciones y en que consis-

ten los llamados semánticos mencionados.

El examen de una condicional inicia procesando a la expresión de la misma, a partir del llamado 5033 que además inicializa a las pilas de operandos y operadores y deposita el valor que tenga el tope al stack de memoria en la variable auxiliar AUXTOP1, de la misma forma y con el mismo propósito que en la proposición de asignación, que ya se vió anteriormente. El llamado 5034 finaliza el procesamiento de la expresión dejando en el stack de operandos el resultado de ella. El siguiente llamado que entra en acción es el 5044, que genera un llamado a la rutina EVALCOND que tomará a la variable o temporal resultante de la expresión, lo evaluará y finalmente dejará en la variable lógica CONDRES la interpretación de la expresión.

EVALCOND puede tomar un escalar real o un arreglo de reales. Si la estructura dispuesta no es ninguna de éstas, se emite un error y se deja a CONDRES como falsa. En caso de un escalar real CONDRES será verdadera si la expresión es distinta de cero, mientras que en un arreglo de reales, sólo si todos sus elementos no son cero, CONDRES será verdadera. Así pues, se pueden escribir condicionales como la siguiente: SI A = B ENTONCES..., donde A y B son dos arreglos multidimensionales de reales.

La forma en que EVALCOND recibe al operando, es por medio de dos parámetros (tipo, localidad), siguiendo el mismo esquema que en las operaciones unarias y binarias (OPERUNA y OPERBIN).

EVALCOND además reestablece el tope al stack de memoria mediante AUXTOP1, liberando así todo el espacio dispuesto para evaluar a la expresión de la condición.

Por último, el llamado 5044 genera al código la línea "IF CONDRES THEN BEGIN" y mete al stack de proposiciones un 1 o un 2, según se tenga respectivamente, una condicional sencilla o una compuesta, lo que se detecta viendo cual es la proposición incondicional (<P INC>) consecuente.

Cuando se llega al final de una proposición ya sea por un punto y como (';') o por el final de una proposición compuesta o blo-

que ("FIN"), puede estar en el stack de proposiciones una condicional sencilla (1) o una alternativa sencilla de condicional (4), de tal manera que hay que emitir al programa objeto la línea "END;", que cierre la proposición compuesta de la condicional (IF CONDRES THEN BEGIN) o de la alternativa de la condicional (END ELSE BEGIN)*, que previamente se ha generado.

Aún más, se puede llegar al caso de tener anidadas varias alternativas sencillas de condicional e inclusive hasta una condicional sencilla, como ocurre en el ejemplo de la figura 6.24, donde se muestra la configuración que tendría el stack de proposiciones.

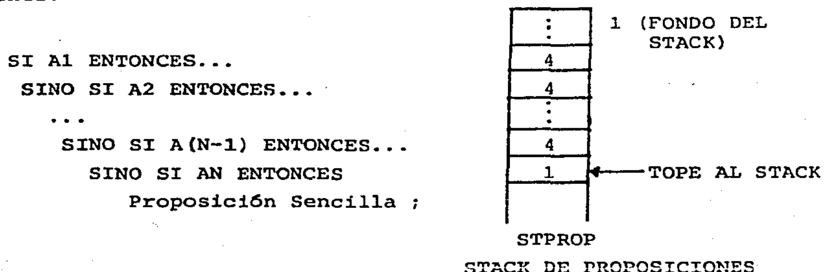


Figura 6.24 Ejemplo del empleo de proposiciones condicionales y del stack de proposiciones.

En este caso, el final de las proposiciones está dado por un punto y coma.

El primer 4 del stack corresponde a la alternativa de la proposición de A1, el último 4 corresponde a la de A(N-1) y el 1 a la condicional sencilla de AN que aparece al final de la anidación.

Por cada uno de estos elementos que hay en el stack se debe generar una línea de "END;" y eliminar dicho elemento del stack.

---

* Posteriormente se verá como y en donde se genera esta línea.

En la figura 6.25 se muestra parte del código generado para el ejemplo de la figura 6.24 .

```
{código para evaluar A1}
IF CONDRES THEN BEGIN
:
END ELSE BEGIN (* ALTERNATIVA DE A1 *)
  {código para evaluar A2}
  IF CONDRES THEN BEGIN
  :
  END ELSE BEGIN (* ALTERNATIVA DE A(N-2)*)
      {código para evaluar A(N-1)}
      IF CONDRES THEN BEGIN
      :
      END ELSE BEGIN (* ALTERNATIVA DE A(N-1)*)
        {código para evaluar AN}
        IF CONDRES THEN BEGIN
            {código para la proposición sencilla }
        END; (* Si AN *)
      END; (* SINO A(N-1)*)
  END; (* SINO A(N-2)*)
  :
END; (* SINO A1 *)
```

Figura 6.25 Parte del Código que se generaría para el Ejemplo de la Figura 6.24

Cuando el final de la proposición o proposiciones anidadas está dado por un punto y coma, implica que también se llegó al final del árbol sintáctico. Es por ésto que a cada línea del programa fuente que corresponda a una proposición, el árbol sintáctico respectivo contiene al final, el llamado semántico 5047 que efectúa todas las acciones mencionadas.

Si el final de la proposición se detecta por medio de un "FIN",

entonces corresponde al llamado 5045 llevar a cabo dichas operaciones.

La rutina que se encarga de revisar el stack de proposiciones, de actualizarlo y emitir código se llama TERCONDICIONAL.

Veamos ahora cual es el papel del llamado 5045, que se invoca al hallar un "FIN" en la frase sintáctica.

Ese "FIN" puede corresponder al término de la primera parte de una condicional con alternativa (...FIN SINO...), a una condicional compuesta que carece de alternativa (SI <Exp> ENTONCES INICIO...FIN;), al fin de una alternativa compuesta (SINO INICIO...FIN), al término de otro tipo de proposición, como por ejemplo una iteración o al fin de un bloque o inclusive, al fin del programa.

En cualquier caso, puede haber en el stack una anidación de alternativas sencillas de condicional, por lo que lo primero que hace el llamado 5045 es ejecutar la rutina TERCONDICIONAL para concluir todas las proposiciones sencillas pendientes.

Si se tiene el fin de la primera parte de una condicional con alternativa, entonces en el stack debe haber una condicional compuesta (2), en cuyo caso sustituyo en el stack, el 2 por un 3, a fin de que cuando se llegue a la parte del "SINO", se pueda comprobar que se terminó de definir la primera parte de la condicional y se detecten correctamente casos erróneos como el siguiente:

```
SI A ENTONCES INICIO
   B : = C;
SINO...
```

en este ejemplo al llegar al "SINO", se encontrará en el tope del stack un 2 que significa que no ha concluido la primera parte de la condicional y sin embargo, ya aparece una alternativa de condicional.

Si por el contrario, al llegar al fin de la primera parte, se encuentra otra cosa en el stack o éste está vacío, entonces hay un error y se cancela el análisis semántico de la frase. Por

ejemplo, si el stack contiene un 5 (alternativa compuesta de condicional) entonces hay dos alternativas sucesivas: ...SINO INICIO...FIN SINO..., lo cual es evidentemente erróneo.

Si la instrucción de "FIN" no corresponde a la situación anterior y en el tope del stack hay un 2 (condicional compuesta) o un 5(alternativa compuesta de condicional), se elimina dicho elemento del stack y se genera al programa objeto: END; . Pero si el stack de proposiciones se encuentra vacío, entonces lo que asume para el "FIN" (mientras no se implanten otras proposiciones), es el fin de una rutina (si el nivel no es uno) o el fin del programa (si el nivel lexicográfico es uno). En este último caso se llama a la rutina BFINSEMANTICA que hace una serie de validaciones, como que no queden etiquetas sin definir, se termina de generar el programa objeto, se cierran archivos y se da por concluido el análisis semántico.

Debido a este manejo de las proposiciones, si se llega al final físico del programa fuente sin haber encontrado el "FIN" del mismo, se marca el error correspondiente y también se llama a BFINSEMANTICA.

Pasemos ahora a ver cual es el manejo de la alternativa de condicional, que es controlada por el llamado semántico 5046.

En primer lugar se determina si la alternativa comprende a una proposición sencilla (4-alternativa sencilla) o a una compuesta (5-alternativa compuesta), por medio del contenido de <P Gral>. A continuación e independientemente del tipo de alternativa, en el stack se debe encontrar una condicional sencilla (1) o el fin de la primera parte de una condicional compuesta (3). Si se encuentra un elemento distinto o el stack está vacío, entonces se indica el error y se cancela el análisis de la proposición. En caso correcto se emite al programa objeto la línea "END ELSE BEGIN" y se reemplaza el tope del stack (1 ó 3) por el tipo de alternativa (4 ó 5).

Al llegar a un punto y coma o a una instrucción de "FIN" y encontrar en el tope del stack un 4 ó un 5 respectivamente, se

genera la línea "END;" que cierra la proposición compuesta de la alternativa en el programa objeto y se saca dicho elemento del stack, tal y como se explicó en los llamados 5047 y 5045 respectivamente.

El stack de proposiciones (STPROP), como ya lo mencione es un mecanismo general que se puede utilizar en la compilación de las demás proposiciones que faltan por implantar, como son las iteraciones y la selección. Así mismo el llamado semántico 5045 se puede utilizar para detectar y procesar la terminación de cualquier proposición compuesta y el llamado 5047 se puede utilizar para detectar cualquier conjunto de proposiciones sencillas que queden anidadas.

Veamos como puede implantarse la iteración equivalente al "WHILE" que se define así:

REPITE MIENTRAS <Exp> ESTO <Proposición Incondicional>

Al hallar el inicio de la iteración, se genera una etiqueta* al código, se procesa la expresión y se genera una proposición condicional igual a la que se genera en la compilación de la condicional.

El código generado entonces consiste de:

```
<ETIQUETA>:
{código para evaluar <Exp>}
IF CONDRES THEN BEGIN
```

Además en el stack de proposiciones se inserta un elemento "mientras sencillo" o un "mientras compuesto".

Mediante los llamados 5045 y 5047 se puede detectar el "FIN" de un "mientras compuesto" o la conclusión de un "mientras sencillo" respectivamente y también se puede detectar un error de semántica.

Finalmente se saca el elemento correspondiente del stack de proposiciones y se genera el código:

```
GOTO <ETIQUETA>;
END; (* IF QUE SIMULA UN MIENTRAS *)
```

* En la siguiente sección se habla detalladamente de como se manipulan y generan etiquetas.

119

De esta forma, para implantar el "mientras" se pueden aprovechar muchas estructuras diseñadas para la condicional.

A continuación se presenta un programa que ejemplifica el empleo de la proposición condicional en los distintos contextos que se han mencionado. Asimismo se muestra la reseña del análisis semántico, el programa objeto generado y finalmente la ejecución de dicho programa objeto.

```
   % PROGRAMA QUE MUESTRA EL EMPLEO DE PROPOSICIONES CONDICIONALES ;
$$D+
INICIO
  REAL A, I, J ;
  ARREGLO X[1:10];
$$D-

  I:=100,    J:=10;
  A:=1;
$$D+
  SI I ENTONCES INICIO
    SI J = O ENTONCES
      A:=0
    SINO INICIO
    $$D
      A:=-1;
      X:=[ 9 : 9, 100 , 1];
    $$D+
    FIN;
  FIN SINO
    I:=-10;

  SI NO A ENTONCES  SINO
    SI O ENTONCES  SINO
      SI X ENTONCES  SINO
        SI J ENTONCES
          I:=55;
FIN;
```

Figura 6.26 Programa que Muestra las Proposiciones Condicionales.

Figura 6.27 Reseña Semántica del Programa de la Figura 6.26

```
PONNIV  :   ARVDT 45 ARDIS 1 INFOLARDIS] 1
DESCRI  :  ARVDT 45 ARDES 0 CP 1 CS 4 CT 1 CC 0
PONNIV  :   ARVDT 46 ARDIS 2 INFOLARDIS] 6
DESCRI  :  ARVDT 46 ARDES 5 CP 1 CS 4 CT 2 CC 0
PONNIV  :   ARVDT 47 ARDIS 3 INFOLARDIS] 11
DESCRI  :  ARVDT 47 ARDES 10 CP 1 CS 4 CT 3 CC 0
PONNIV  :   ARVDT 48 ARDIS 4 INFOLARDIS] 16
DESCRI  :  ARVDT 48 ARDES 15 CP 1 CS 17 CT 4 CC 1
DESCRI  :  ARVDT 0 ARDES 20 CP 1 CS 10 CT 0 CC 0
```

Declaración de Variables.

```
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
VE DESCRIPTOR     : DIRVDT 46 LIGA 2 INFO 6
 DESCRIPTOR DE   46 :  CP 1 CS 4 CT 2 CC 0
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. 2 NIV 1
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
```

Análisis de la Expresión: I

```
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. 2 NIV 1
CONDICIONAL COMPUESTA
INSERCION AL STACK DE PROPOSICIONES: IND 1 PROP. 2
TOPE DEL STACK DE PROPOSICIONES: IND 1 PROP. 2
```

SI I ENTONCES INICIO Condicional Compuesta C2)

```
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
VE DESCRIPTOR     : DIRVDT 47 LIGA 3 INFO 11
 DESCRIPTOR DE   47 :  CP 1 CS 4 CT 3 CC 0
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. 3 NIV 1
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MASUN
METE OPERADOR :   IOPDOR 0 OPERACION IGUAL
METE OPERADOR :   IOPDOR 1 OPERACION MASUN
METE OPERANDO :   IOPNDO 1 TIPO 1 LOC.   0.00000
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 1 OPERACION MASUN
OPERACION BINARIA
SACA OPERADOR :   IOPDOR 0 OPERACION IGUAL
SACA OPERANDO :   IOPNDO 1 TIPO 1 LOC.   0.00000
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. 3 NIV 1
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. -1 NIV 7885
FIN DE EXPRESION ARITMETICA
```

Análisis de la Expresión: J=0

```
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. -1 NIV 7885
CONDICIONAL SENCILLA
INSERCION AL STACK DE PROPOSICIONES: IND 2 PROP. 1
```

SI J=0 ENTONCES Condicional Sencilla C1)

```
INICIO DE ASIGNACION. ANALISIS DEL LADO DERECHO
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
METE OPERANDO :   IOPNDO 0 TIPO 1 LOC.   0.00000
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
ANALISIS DE LA PARTE IZQUIERDA DE LA ASIGNACION
VE DESCRIPTOR     : DIRVDT 45 LIGA 1 INFO 1
 DESCRIPTOR DE   45 :  CP 1 CS 4 CT 1 CC 0
METE OPERANDO :   IOPNDO 1 TIPO 2 LOC. 1 NIV 1
SACA OPERANDO :   IOPNDO 1 TIPO 2 LOC. 1 NIV 1
SACA OPERANDO :   IOPNDO 0 TIPO 1 LOC.   0.00000
FIN DE ASIGNACION
```

Asignación: A:=0;

```
ALTERNATIVA COMPUESTA DE CONDICIONAL
TOPE DEL STACK DE PROPOSICIONES: IND 2 PROP. 1
SALE ELEMENTO DEL STACK DE PROPOSICIONES: IND 1
INSERCION AL STACK DE PROPOSICIONES: IND 2 PROP. 5
TOPE DEL STACK DE PROPOSICIONES: IND 2 PROP. 5
TOPE DEL STACK DE PROPOSICIONES: IND 2 PROP. 5
```

SINO INICIO
SALE SI (Cond. Sencilla)
ENTRA SI (Alternativa Compuesta).

Figura 6.27 Reseña Semántica del Programa de la Figura 6.26

```
PONNIV :   ARVDT 45 ARDIS 1 INFOLARDIS] 1
DESCRI   : ARVDT 45 ARDES 0 CP 1 CS 4 CT 1 CC 0
PONNIV :   ARVDT 46 ARDIS 2 INFOLARDIS] 6
DESCRI   : ARVDT 46 ARDES 5 CP 1 CS 4 CT 2 CC 0
PONNIV :   ARVDT 47 ARDIS 3 INFOLARDIS] 11
DESCRI   : ARVDT 47 ARDES 10 CP 1 CS 4 CT 3 CC 0
PONNIV :   ARVDT 48 ARDIS 4 INFOLARDIS] 16
DESCRI   : ARVDT 48 ARDES 15 CP 1 CS 17 CT 4 CC 1
DESCRI   : ARVDT 0 ARDES 20 CP 1 CS 10 CT 0 CC 0
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
VE DESCRIPTOR    : DIRVDT 46 LIGA 2 INFO 6
 DESCRIPTOR DE   46 :  CP 1 CS 4 CT 2 CC 0
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. 2 NIV 1
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. 2 NIV 1
CONDICIONAL COMPUESTA
INSERCION AL STACK DE PROPOSICIONES: IND 1 PROP. 2
TOPE DEL STACK DE PROPOSICIONES: IND 1 PROP. 2
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
VE DESCRIPTOR    : DIRVDT 47 LIGA 3 INFO 11
 DESCRIPTOR DE   47 :  CP 1 CS 4 CT 3 CC 0
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. 3 NIV 1
OPERACION UNARIA.
SACA OPERADOR :   IOPDOR 0 OPERACION MASUN
METE OPERADOR :   IOPDOR 0 OPERACION IGUAL
METE OPERADOR :   IOPDOR 1 OPERACION MASUN
METE OPERANDO :   IOPNDO 1 TIPO 1 LOC.   0.00000
OPERACION UNARIA.
SACA OPERADOR :   IOPDOR 1 OPERACION MASUN
OPERACION BINARIA
SACA OPERADOR :   IOPDOR 0 OPERACION IGUAL
SACA OPERANDO :   IOPNDO 1 TIPO 1 LOC.   0.00000
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. 3 NIV 1
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. -1 NIV 7885
FIN DE EXPRESION ARITMETICA
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. -1 NIV 7885
CONDICIONAL SENCILLA
INSERCION AL STACK DE PROPOSICIONES: IND 2 PROP. 1
INICIO DE ASIGNACION. ANALISIS DEL LADO DERECHO
INICIO DE EXPRESION ARITMETICA.
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
METE OPERANDO :   IOPNDO 0 TIPO 1 LOC.   0.00000
OPERACION UNARIA.
SACA OPERADOR :   IOPDOR 0 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
ANALISIS DE LA PARTE IZQUIERDA DE LA ASIGNACION
VE DESCRIPTOR    : DIRVDT 45 LIGA 1 INFO 1
 DESCRIPTOR DE   45 :  CP 1 CS 4 CT 1 CC 0
METE OPERANDO :   IOPNDO 1 TIPO 2 LOC. 1 NIV 1
SACA OPERANDO :   IOPNDO 1 TIPO 2 LOC. 1 NIV 1
SACA OPERANDO :   IOPNDO 0 TIPO 1 LOC.   0.00000
FIN DE ASIGNACION
ALTERNATIVA COMPUESTA DE CONDICIONAL
TOPE DEL STACK DE PROPOSICIONES: IND 2 PROP. 1
SALE ELEMENTO DEL STACK DE PROPOSICIONES: IND 1
INSERCION AL STACK DE PROPOSICIONES: IND 2 PROP. 5
TOPE DEL STACK DE PROPOSICIONES: IND 2 PROP. 5
TOPE DEL STACK DE PROPOSICIONES: IND 2 PROP. 5
```

Handwritten annotations (right margin, bracketing the listing):

- Declaración de Variables.
- Análisis de la Expresión: I
- SI I ENTONCES Inicio condicional Compuesta C2)
- Análisis de la Expresión: J=0
- SI J=0 ENTONCES Condicional Sencilla C1)
- Asignación: A := 0;
- SINO INICIO SALE 1 (Cond. Sencilla) ENTRA 5 (Alternativa Compuesta).

```
TOPE DEL STACK DE PROPOSICIONES: IND 2 PROP. 5
SALE ELEMENTO DEL STACK DE PROPOSICIONES: IND 1
FIN DE ALTERNATIVA COMPUESTA DE CONDICIONAL
TOPE DEL STACK DE PROPOSICIONES: IND 1 PROP. 2
TOPE DEL STACK DE PROPOSICIONES: IND 1 PROP. 2
TOPE DEL STACK DE PROPOSICIONES: IND 1 PROP. 2
SALE ELEMENTO DEL STACK DE PROPOSICIONES: IND 0
INSERCION AL STACK DE PROPOSICIONES: IND 1 PROP. 3
FIN DE LA 1A. PARTE DE CONDICIONAL COMPUESTA
ALTERNATIVA SENCILLA DE CONDICIONAL
TOPE DEL STACK DE PROPOSICIONES: IND 1 PROP. 3
SALE ELEMENTO DEL STACK DE PROPOSICIONES: IND 0
INSERCION AL STACK DE PROPOSICIONES: IND 1 PROP. 4
INICIO DE ASIGNACION. ANALISIS DEL LADO DERECHO
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MENOSUN
METE OPERANDO :   IOPNDO 0 TIPO 1 LOC.  1.00000E1
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MENOSUN
SACA OPERANDO :   IOPNDO 0 TIPO 1 LOC.  1.00000E1
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. -1 NIV 7885
FIN DE EXPRESION ARITMETICA
ANALISIS DE LA PARTE IZQUIERDA DE LA ASIGNACION
VE DESCRIPTOR    : DIRVDT 46 LIGA 2 INFO 6
 DESCRIPTOR DE   46 :  CP 1 CS 4 CT 2 CC 0
METE OPERANDO :   IOPNDO 1 TIPO 2 LOC. 2 NIV 1
SACA OPERANDO :   IOPNDO 1 TIPO 2 LOC. 2 NIV 1
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. -1 NIV 7885
FIN DE ASIGNACION
TOPE DEL STACK DE PROPOSICIONES: IND 1 PROP. 4
SALE ELEMENTO DEL STACK DE PROPOSICIONES: IND 0
TOPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
VE DESCRIPTOR    : DIRVDT 45 LIGA 1 INFO 1
 DESCRIPTOR DE   45 :  CP 1 CS 4 CT 1 CC 0
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. 1 NIV 1
METE OPERADOR :   IOPDOR 1 OPERACION NEGACION
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 1 OPERACION NEGACION
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. 1 NIV 1
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. -1 NIV 7885
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. -1 NIV 7885
CONDICIONAL SENCILLA
INSERCION AL STACK DE PROPOSICIONES: IND 1 PROP. 1
ALTERNATIVA SENCILLA DE CONDICIONAL
TOPE DEL STACK DE PROPOSICIONES: IND 1 PROP. 1
SALE ELEMENTO DEL STACK DE PROPOSICIONES: IND 0
INSERCION AL STACK DE PROPOSICIONES: IND 1 PROP. 4
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
METE OPERANDO :   IOPNDO 0 TIPO 1 LOC.  0.00000
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
SACA OPERANDO :   IOPNDO 0 TIPO 1 LOC.  0.00000
CONDICIONAL SENCILLA
INSERCION AL STACK DE PROPOSICIONES: IND 2 PROP. 1
ALTERNATIVA SENCILLA DE CONDICIONAL
TOPE DEL STACK DE PROPOSICIONES: IND 2 PROP. 1
SALE ELEMENTO DEL STACK DE PROPOSICIONES: IND 1
INSERCION AL STACK DE PROPOSICIONES: IND 2 PROP. 4
```

Handwritten annotations:

- FIN; Termina Condicional. SALE 5 (Alternativa compuesta)
- FIN SINO. SALE 2 (Condicional Comp) ENTRA 3 (Fin de 1ª Parte de Condicional)
- ...SINO... SALE 3. ENTRA 4 (Alternativa Sencilla)
- I:=-10;
- Llamado 5047. Sale 4. condicional-alternativa Sencilla.
- Expresión: NO A
- SI NO A ENTONCES Condicional Sencilla
- SINO. Alternativa Sencilla
- Expresión: 0
- SI 0 ENTONCES Condicional Sencilla
- SINO Alternativa Sencilla

```
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR O OPERACION MASUN
VE DESCRIPTOR    : DIRVDT 48 LIGA 4 INFO 16
 DESCRIPTOR DE   48 :   CP 1 CS 17 CT 4 CC 1
METE OPERANDO :   IOPNDO O TIPO 4 LOC. 4 NIV 1
OPERACION UNARIA
SACA OPERADOR :   IOPDOR O OPERACION MASUN
FIN DE EXPRESION ARITMETICA
SACA OPERANDO :   IOPNDO O TIPO 4 LOC. 4 NIV 1
CONDICIONAL SENCILLA
INSERCION AL STACK DE PROPOSICIONES: IND 3 PROP. 1
ALTERNATIVA SENCILLA DE CONDICIONAL
TOPE DEL STACK DE PROPOSICIONES: IND 3 PROP. 1
SALE ELEMENTO DEL STACK DE PROPOSICIONES: IND 2
INSERCION AL STACK DE PROPOSICIONES: IND 3 PROP. 4
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR O OPERACION MASUN
VE DESCRIPTOR    : DIRVDT 47 LIGA 3 INFO 11
 DESCRIPTOR DE   47 :   CP 1 CS 4 CT 3 CC 0
METE OPERANDO :   IOPNDO O TIPO 2 LOC. 3 NIV 1
OPERACION UNARIA
SACA OPERADOR :   IOPDOR O OPERACION MASUN
FIN DE EXPRESION ARITMETICA
SACA OPERANDO :   IOPNDO O TIPO 2 LOC. 3 NIV 1
CONDICIONAL SENCILLA
INSERCION AL STACK DE PROPOSICIONES: IND 4 PROP. 1
INICIO DE ASIGNACION. ANALISIS DEL LADO DERECHO
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR O OPERACION MASUN
METE OPERANDO :   IOPNDO O TIPO 1 LOC.   5.50000E1
OPERACION UNARIA
SACA OPERADOR :   IOPDOR O OPERACION MASUN
FIN DE EXPRESION ARITMETICA
ANALISIS DE LA PARTE IZQUIERDA DE LA ASIGNACION
VE DESCRIPTOR    : DIRVDT 46 LIGA 2 INFO 6
 DESCRIPTOR DE   46 :   CP 1 CS 4 CT 2 CC 0
METE OPERANDO :   IOPNDO 1 TIPO 2 LOC. 2 NIV 1
SACA OPERANDO :   IOPNDO 1 TIPO 2 LOC. 2 NIV 1
SACA OPERANDO :   IOPNDO O TIPO 1 LOC.   5.50000E1
FIN DE ASIGNACION
TOPE DEL STACK DE PROPOSICIONES: IND 4 PROP. 1
SALE ELEMENTO DEL STACK DE PROPOSICIONES: IND 3
TOPE DEL STACK DE PROPOSICIONES: IND 3 PROP. 4
SALE ELEMENTO DEL STACK DE PROPOSICIONES: IND 2
TOPE DEL STACK DE PROPOSICIONES: IND 2 PROP. 4
SALE ELEMENTO DEL STACK DE PROPOSICIONES: IND 1
TOPE DEL STACK DE PROPOSICIONES: IND 1 PROP. 4
SALE ELEMENTO DEL STACK DE PROPOSICIONES: IND 0
TOPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
TOPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
TOPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
MINIMA MEMORIA DISPONIBLE : 3993
**** ACABO LA COMPILACION ***  0 ERRORES **
```

Handwritten annotations (bracketed to the right of the listing):

- Expresión: X
- SI X ENTONCES
- SINO
- Expresión: ∅
- SI ∅ ENTONCES Condicional Sencilla
- I:= 55;
- Sale condicional Sencilla y alternativas Sencillas.
- AL FINAL EL STACK DE PROPOSICIONES QUEDA VACIO. SIN ERROR.

Figura 6.28 Programa Objeto Correspondiente al Ejemplo de la Figura 6.26

```
(*$S++*)
(*$G+*)
PROGRAM LENGES1;
USES TRANSCEN, CODGRAL, INICOD;
LABEL 1, 2, 3, 4, 5, 6, 7, 8, 9, 10;
PROCEDURE INI11;
BEGIN
INICIALI; <————— Inicialización del Programa.
NIVEL:=1;
DECIARR(4,1,10); <————— Asigna Descripción Dinámica al Arreglo X
TOPEMEM:=DISPLAY[1]+16;
END; (* INI11 *)
BEGIN
INI11;
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)  ┐
SELEC[0]:=0;                                            ├ I:=100;
MUEVE(2,2,1, 1.00000E2,1);                              │
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)              ┘
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)  ┐
SELEC[0]:=0;                                            ├ J:=10;
MUEVE(3,2,1, 1.00000E1,1);                              │
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)              ┘
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)  ┐
SELEC[0]:=0;                                            ├ A:=1;
MUEVE(1,2,1, 1.00000,1);                                │
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)              ┘
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)  ┐ Evaluación de Expresión:
     (* TERMINO EL PROCESAMIENTO DE LA EXPRESION ***)   │ I.
NIVELUNO:=1;                                            │ SI I ENTONCES...
EVALCOND(2,2);                                          │
IF CONDRES THEN BEGIN                                   ┘
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)  ┐
NIVELUNO:=1;                                            │ Evaluación de:
OPERBIN(IGUAL,3,2, 0.00000,1);                          ├ J=0
     (* TERMINO EL PROCESAMIENTO DE LA EXPRESION ***)   │ SI J=0 ENTONCES...
EVALCOND(-1,2);                                         │
IF CONDRES THEN BEGIN                                   ┘
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)  ┐
SELEC[0]:=0;                                            ├ A:=0;
MUEVE(1,2,1, 0.00000,1);                                │
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)              ┘
END ELSE BEGIN <————— SINO J=0
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)  ┐
OPERUNA(MENOSUN, 1.00000,1);                            │
SELEC[0]:=0;                                            │
MUEVE(1,2,1,-1,2);                                      │
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)              │ A:= -1;
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)  ├ y
ARREXP[1]:=9;                                           │ X:= I[9:9(100)]]
AXARREXP( 1.00000E2,2);                                 │
BXARREXP(4,1,11,9);                                     │
ARREXP[3]:=10000;                                       │
CONSARREXP(3);                                          │
SELEC[0]:=0;                                            │
MUEVE(4,4,1,-1,4);                                      │
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)              ┘
END; (* SINO *) <————— termina: SINO J=0
END ELSE BEGIN <————— SINO I
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)  ┐
OPERUNA(MENOSUN, 1.00000E1,1);                          ├ I:=-10;
SELEC[0]:=0;                                            │
MUEVE(2,2,1,-1,2);
```

```
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
END; (* SINO *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
NIVELUNO:=1;
OPERUNA(NEGACION,1,2);
     (* TERMINO EL PROCESAMIENTO DE LA EXPRESION ***)
EVALCOND(-1,2);
IF CONDRES THEN BEGIN
END ELSE BEGIN
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
IF FALSE THEN BEGIN
END ELSE BEGIN
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
     (* TERMINO EL PROCESAMIENTO DE LA EXPRESION ***)
NIVELUNO:=1;
EVALCOND(4,4);
IF CONDRES THEN BEGIN
END ELSE BEGIN
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
     (* TERMINO EL PROCESAMIENTO DE LA EXPRESION ***)
NIVELUNO:=1;
EVALCOND(3,2);
IF CONDRES THEN BEGIN
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
SELEC[0]:=0;
MUEVE(2,2,1, 5.50000E1,1);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
END; (* SI *)
END; (* SINO *)
END; (* SINO *)
END; (* SINO *)
CLOSE(LST,LOCK);
END.
```

Handwritten annotations (right margin, in order):

- termina SINO I
- Evaluación de: NO A / SI NO A ENTONCES...
- SINO: NO A.
- SI O ENTONCES SINO...
- Evaluación de X. / SI X ENTONCES SINO...
- Evaluación de J. / SI J ENTONCES...
- I:= 55;
- Terminan condicionales.

Figura 6.29 Ejecución del Programa Objeto de la Figura 6.28

```
RESOPNDO   1.00000E2 1 0 1 1
COMOPNDO   1.00000E2
RESOPNDO   2.00000 2 2 2 1
********* MUEVE *** OPNDO1 2 2 OPNDO2 1 0VALOR  1.00000E2
RESOPNDO   1.00000E1 1 0 1 1
COMOPNDO   1.00000E1
RESOPNDO   3.00000 2 3 2 1
********* MUEVE *** OPNDO1 2 3 OPNDO2 1 0VALOR  1.00000E1
RESOPNDO   1.00000 1 0 1 1
COMOPNDO   1.00000
RESOPNDO   1.00000 2 1 2 1
********* MUEVE *** OPNDO1 2 1 OPNDO2 1 0VALOR  1.00000
RESOPNDO   2.00000 2 2 2 1
COMOPNDO   1.00000E2
RESOPNDO   0.00000 1 0 1 1
COMOPNDO   0.00000
RESOPNDO   3.00000 2 3 2 1
COMOPNDO   1.00000E1
OP. BIN. 3 OPNDO1 2 3  1.00000E1 OPNDO2 1 0  0.00000 RES 2 16  0.00000
METE DES. IND 0 TIPO 2 LOC. 16
LONG.:  TIPO 2 LOC. 16 LONGITUD 1
RESOPNDO -1.00000 2 SACA DES. IND 0 TIPO 2 LOC. 16
16 2 1
COMOPNDO   0.00000
RESOPNDO   1.00000 1 0 1 1
COMOPNDO   1.00000
OP. UNA. 14 OPNDO1 1 0  1.00000 RES 2 16 -1.00000
METE DES. IND 0 TIPO 2 LOC. 16
LONG.:  TIPO 2 LOC. 16 LONGITUD 1
RESOPNDO -1.00000 2 SACA DES. IND 0 TIPO 2 LOC. 16
16 2 1
COMOPNDO -1.00000
RESOPNDO   1.00000 2 1 2 1
********* MUEVE *** OPNDO1 2 1 OPNDO2 2 16VALOR -1.00000
METE DES. IND 0 TIPO 2 LOC. 16
METE DES. IND 1 TIPO 4 LOC. 17
APAE FINAL 4
ARREXP : I 0        3 *
I 1      9 * I 2      -1 * I 3   10000 *
TOPEMEM= 28
RESOPNDO -1.00000 4 SACA DES. IND 0 TIPO 4 LOC. 17
17 4 0
RESOPNDO   4.00000 4 4 4 0
********* MUEVE *** OPNDO1 4 4 OPNDO2 4 17
RESOPNDO   1.00000 2 1 2 1
COMOPNDO -1.00000
OP. UNA. 15 OPNDO1 2 1 -1.00000 RES 2 16  0.00000
METE DES. IND 0 TIPO 2 LOC. 16
LONG.:  TIPO 2 LOC. 16 LONGITUD 1
RESOPNDO -1.00000 2 SACA DES. IND 0 TIPO 2 LOC. 16
16 2 1
COMOPNDO   0.00000
RESOPNDO   4.00000 4 4 4 0
RESOPNDO   3.00000 2 3 2 1
COMOPNDO   1.00000E1
RESOPNDO   5.50000E1 1 0 1 1
COMOPNDO   5.50000E1
RESOPNDO   2.00000 2 2 2 1
********* MUEVE *** OPNDO1 2 2 OPNDO2 1 0VALOR  5.50000E1
```

Anotaciones manuscritas:

- I := 100;
- 1 LOC / TIPO; 2 LOC / TIPO
- J := 10;
- ESCALAR / TIPO / LOC
- A := 1;
- I vale 100
- LOC ANTES DE PROCESAR OPERANDO
- TIPO ANTES DE DECODIFICAR OPERANDO.
- J = 0 es FALSO (0)
- A := -1;
- X := [[9 : 9 (100)]];
- 0 es FALSO.
- NO A es FALSO
- Operando X. Como contiene un 0 => es FALSO. J no es CERO => como se toma como VERDADERO (1).
- I := 55;

H) PROPOSICION DE TRANSFERENCIA INCONDICIONAL

Esta proposición equivale al "GOTO" de PASCAL. Al igual que en aquel lenguaje, las etiquetas son numéricas, pero a diferencia de aquel, las etiquetas no se declaran y son estrictamente locales al bloque en que se encuentren.

Para implantar esta proposición en el código se decidió emplear el "GOTO" de PASCAL, dado que es muy semejante en su forma de utilizar y en su propósito con la proposición VER del lenguaje estadístico.

Sin embargo, como se mencionó hace un momento, PASCAL requiere que las etiquetas empleadas sean declaradas. Debido a ello, en cada bloque o rutina del programa objeto se declara un conjunto de etiquetas, que va de la etiqueta 1 hasta la etiqueta MAXET, siendo esta una constante perteneciente al analizador semántico, que representa el máximo número de etiquetas que pueden aparecer por bloque en el programa fuente.

Por el momento se permiten hasta diez etiquetas por bloque y como todavía no se implantan rutinas, las etiquetas se declaran sólo para el programa principal.

La línea generada es "LABEL 1,2,3,4,5,6,7,8,9,10;" y es emitida por la rutina ININIVEL.

En el PASCAL utilizado* en la presente implantación del código, si hay etiquetas declaradas, pero que no aparecen, ni son utilizadas por proposición de transferencia alguna, no acarrean error en la compilación, ni en la ejecución del programa objeto. Pero si ocurre que el compilador sea transportado a otra versión de PASCAL, donde esa situación provoque errores, basta con que al final de cada bloque del programa objeto, se generen proposiciones vacias antecedidas por las etiquetas no aparecidas en el bloque.

---

* La versión empleada es el UCSD PASCAL para APPLE.

Por ejemplo, si se emplean 6 etiquetas y las 4 restantes ni siquiera aparecen, antes de generar el "END;" del bloque se genera la siguiente línea:

7:; 8:; 9:; 10:;

que contiene proposiciones legales según la definición original de PASCAL (ver Jensen y Wirth [J1]).

Por otra parte, como se exige en el programa fuente, que las etiquetas sean estrictamente locales al bloque en que aparezcan, no hay problema al declarar el mismo conjunto de etiquetas en cada bloque.

Para auxiliar el manejo de las etiquetas y transferencias que aparecen en el programa fuente, se diseñó una estructura de datos que se presenta a continuación:

```
VAR TGOTO: ARRAY [1.. TAMTGOTO] OF RECORD
                NOM: INTEGER;
                VAL: 0..3;
              END;
```

TAMTGOTO es una constante.

Cada elemento de TGOTO consiste de dos componentes: NOM, que es el nombre de la etiqueta en el programa fuente y consiste en un número entero y positivo. VAL es una bandera que indica las condiciones en que ha aparecido dicha etiqueta en el programa fuente y se presentan a continuación:

VAL	SIGNIFICADO
0	La etiqueta no ha aparecido en el programa fuente.
1	La etiqueta ya fué definida, es decir, apareció como encabezado o etiqueta de una proposición.
2	La etiqueta ya fué utilizada en una proposición de transferencia.
3	La etiqueta ya fué tanto definida como utilizada en una transferencia.

TGOTO es un diccionario que se administra en forma lineal y tiene un apuntador, INDGT que siempre señala a la última celda

utilizada.

Como las etiquetas son estrictamente locales al bloque en donde aparecen, entonces todas las transferencias incondicionales deben dirigirse a etiquetas en el mismo bloque y para ser más precisos, ésto debe ocurrir dentro del cuerpo de una misma rutina o dentro del programa principal. Como los cuerpos de los bloques son independientes entre sí, es decir, a la mitad del cuerpo de una rutina no puede aparecer el cuerpo de otra rutina, entonces puedo utilizar a la misma estructura TGOTO para manejar las etiquetas de todos los bloques que haya en el programa fuente, pues al finalizar el cuerpo de un bloque e inicializar otro, la información anterior debe eliminarse y la tabla TGOTO queda inicializada en vacío.

Sea I un índice a TGOTO, entonces TGOTO[I]. NOM es el nombre de la etiqueta en el programa fuente, mientras que I es el nombre de la misma, que le corresponde en el programa objeto. Veamos un ejemplo en la figura 6.30 .

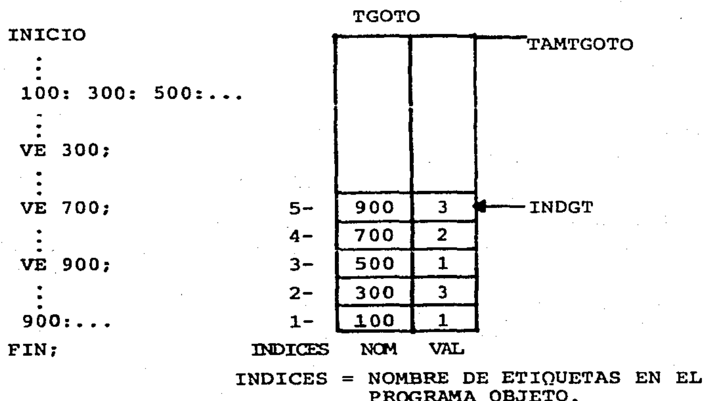


Figura 6.30 Empleo del Diccionario de Etiquetas TGOTO.

En el ejemplo se puede observar como queda el diccionario de etiquetas al finalizar el bloque respectivo. A la etiqueta

100, le corresponde la 1 en el programa objeto y a la 900, la etiqueta 5 en el código. Además puede verse que las etiquetas 100 y 500 sólo se definieron y que la 700 fué utilizada pero no definida en el bloque.

Si al terminar un bloque o el programa se encuentran etiquetas cuyo valor (VAL) es 1, entonces se manda al listado de la semántica un aviso de que las etiquetas fueron definidas pero no utilizadas en ese bloque. Si por el contrario, el valor es 2, entonces la etiqueta fué utilizada pero nunca definida en ese bloque y por tanto se emite el error respectivo.

Para procesar a las etiquetas se dispone de dos llamados semánticos en la gramática y se presentan enseguida:

<P> :: = <Decl> | <L Etiq> <P Gral> <5047> | e

<L Etiq> :: = <5042> <Número> ":"<L Etiq> |
e

<P Ver> :: = "VER" <5043> <Número>

Veamos primeramente que se hace al definirse una o varias etiquetas.

Al igual que en PASCAL, en el lenguaje estadístico cualquier proposición, incluyendo la vacía, puede ser encabezada por etiquetas, pero a diferencia de PASCAL, en donde sólo puede aparecer una etiqueta por proposición, en el lenguaje estadístico puede haber una lista de ellas.

Cada llamado 5042 procesa a una sola etiqueta. Al determinar el nombre de ella, se busca en el diccionario de etiquetas TGOTO y se da de alta si es necesario. Además se actualiza el campo VAL a 1 ó 3, según esté sólo definida o definida y usada, respectivamente. Finalmente se genera la línea: "<ETIQ>:", donde <ETIQ> es el nombre de la etiqueta en el programa objeto. Si en la lista de etiquetas <L Etiq> hay más elementos, entonces se emite una proposición vacía: ";". En esta forma, la última etiqueta generada encabezará a una proposición del programa objeto.

Pasemos a ver como se analiza una transferencia incondicional,

que corresponde al llamado 5043.

En primer lugar se busca la etiqueta del programa fuente en TGOTO, se da de alta si es necesario y se actualiza el campo VAL a 2 y si ya antes fué definida, entonces a 3. Finalmente se genera el brinco: "GOTO <ETIQ>;", donde <ETIQ> es el nombre de la etiqueta en el programa objeto.

Mediante la estructura TGOTO y con la restricción de que las etiquetas son estrictamente locales, es sencillo manejar a las etiquetas y determinar cuando se han empleado erróneamente.

A continuación se presenta un programa que muestra el empleo de la proposición de transferencia incondicional, junto con la reseña semántica de la compilación, el programa objeto generado y la ejecución del mismo.

```
(    1)      % PROGRAMA QUE EJEMPLIFICA LA PROPOSICION DE SALTO INCONDICIONAL ;
(    1) $$D+
(    2) INICIO
(    2)   CONSTANTE LIMITE = 2;
(    3)   REAL I ;
(    4)
(    4)   I:=1;
(    5) 900 : 800 : 500 :
(    5)   SI I < LIMITE ENTONCES INICIO
(    6)   $$D-
(    7)     I:=I+1;
(    8)   $$D+
(    9)     VER 500 ;
(   10)   FIN SINO
(   10)     VER 200 ;
(   11)
(   11) 300 : I:=0;
(   12)
(   12) 200 : SI NO I ENTONCES
(   12)          VER 900;
(   13)
(   13) FIN;
(   14)
(   14)
(   14)
**** ACABO EL ANALIZADOR LEXICO-SINTACTICO ****
     0 ERRORES
```

Figura 6.31 Ejemplo del Empleo de la Proposición de Transferencia Incondicional.

Figura 6.32 Reseña Semántica del Programa de la Figura 6.31

```
PONNIV :   APVDT 45 APDIS 1 INFO[APDIS] 1
DESCRI   : APVDT 45 APDES 0 CP 1 CS 3 CT 267 CC 0
PONNIV :   APVDT 47 APDIS 2 INFO[APDIS] 6
DESCRI   : APVDT 47 APDES 5 CP 1 CS 4 CT 1 CC 0
```

Declaración de Variables

```
INICIO DE ASIGNACION. ANALISIS DEL LADO DERECHO
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
METE OPERANDO :   IOPNDO 0 TIPO 1 LOC.  1.00000
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
ANALISIS DE LA PARTE IZQUIERDA DE LA ASIGNACION
VE DESCRIPTOR    : DIRVDT 47 LIGA 2 INFO 6
 DESCRIPTOR DE   47 :   CP 1 CS 4 CT 1 CC 0
METE OPERANDO :   IOPNDO 1 TIPO 2 LOC. 1 NIV 1
SACA OPERANDO :   IOPNDO 1 TIPO 2 LOC. 1 NIV 1
SACA OPERANDO :   IOPNDO 0 TIPO 1 LOC.  1.00000
FIN DE ASIGNACION
```

I:= 1;

```
TOPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
APARICION DE UNA ETIQUETA
ETIQUETA: INDICE 1  NOMBRE 900
APARICION DE UNA ETIQUETA
ETIQUETA: INDICE 2  NOMBRE 800
APARICION DE UNA ETIQUETA
ETIQUETA: INDICE 3  NOMBRE 500
```

Se definen y se dan de alta etiquetas 900 (nombre interno 1), 800 (2) y 500 (3).

```
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
VE DESCRIPTOR    : DIRVDT 47 LIGA 2 INFO 6
 DESCRIPTOR DE   47 :   CP 1 CS 4 CT 1 CC 0
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. 1 NIV 1
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MASUN
METE OPERADOR :   IOPDOR 0 OPERACION MENOR
METE OPERADOR :   IOPDOR 1 OPERACION MASUN
VE DESCRIPTOR    : DIRVDT 45 LIGA 1 INFO 1
 DESCRIPTOR DE   45 :   CP 1 CS 3 CT 267 CC 0
METE OPERANDO :   IOPNDO 1 TIPO 1 LOC.  2.00000
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 1 OPERACION MASUN
OPERACION BINARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MENOR
SACA OPERANDO :   IOPNDO 1 TIPO 1 LOC.  2.00000
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. 1 NIV 1
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. -1 NIV 7885
FIN DE EXPRESION ARITMETICA
```

Expresión: I < LIMITE

```
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. -1 NIV 7885
CONDICIONAL COMPUESTA
INSERCION AL STACK DE PROPOSICIONES: IND 1 PROP. 2
```

SI I < LIMITE ENTONCES Inicio. Condicional Compuesta

```
TOPE DEL STACK DE PROPOSICIONES: IND 1 PROP. 2
BRINCO
ETIQUETA: INDICE 3  NOMBRE 500
```

VER 500; SE GENERA: GOTO 3;

```
TOPE DEL STACK DE PROPOSICIONES: IND 1 PROP. 2
TOPE DEL STACK DE PROPOSICIONES: IND 1 PROP. 2
TOPE DEL STACK DE PROPOSICIONES: IND 1 PROP. 2
SALE ELEMENTO DEL STACK DE PROPOSICIONES: IND 0
INSERCION AL STACK DE PROPOSICIONES: IND 1 PROP. 3
FIN DE LA 1A. PARTE DE CONDICIONAL COMPUESTA
ALTERNATIVA SENCILLA DE CONDICIONAL
TOPE DEL STACK DE PROPOSICIONES: IND 1 PROP. 3
SALE ELEMENTO DEL STACK DE PROPOSICIONES: IND 0
INSERCION AL STACK DE PROPOSICIONES: IND 1 PROP. 4
```

FIN SINO

```
BRINCO
```

```
ETIQUETA: INDICE 4  NOMBRE 200
TOPE DEL STACK DE PROPOSICIONES: IND 1 PROP. 4
SALE ELEMENTO DEL STACK DE PROPOSICIONES: IND 0
TOPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
APARICION DE UNA ETIQUETA
ETIQUETA: INDICE 5  NOMBRE 300
INICIO DE ASIGNACION. ANALISIS DEL LADO DERECHO
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
METE OPERANDO :   IOPNDO 0 TIPO 1 LOC.   0.00000
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
ANALISIS DE LA PARTE IZQUIERDA DE LA ASIGNACION
VE DESCRIPTOR    : DIRVDT 47 LIGA 2 INFO 6
 DESCRIPTOR DE  47 :   CP 1 CS 4 CT 1 CC 0
METE OPERANDO :   IOPNDO 1 TIPO 2 LOC. 1 NIV 1
SACA OPERANDO :   IOPNDO 1 TIPO 2 LOC. 1 NIV 1
SACA OPERANDO :   IOPNDO 0 TIPO 1 LOC.   0.00000
FIN DE ASIGNACION
TOPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
APARICION DE UNA ETIQUETA
ETIQUETA: INDICE 4  NOMBRE 200
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
VE DESCRIPTOR    : DIRVDT 47 LIGA 2 INFO 6
 DESCRIPTOR DE  47 :   CP 1 CS 4 CT 1 CC 0
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. 1 NIV 1
METE OPERADOR :   IOPDOR 1 OPERACION NEGACION
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 1 OPERACION NEGACION
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. 1 NIV 1
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. -1 NIV 7885
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. -1 NIV 7885
CONDICIONAL SENCILLA
INSERCION AL STACK DE PROPOSICIONES: IND 1 PROP. 1
BRINCO
ETIQUETA: INDICE 1  NOMBRE 900
TOPE DEL STACK DE PROPOSICIONES: IND 1 PROP. 1
SALE ELEMENTO DEL STACK DE PROPOSICIONES: IND 0
TOPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
TOPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
TOPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
MINIMA MEMORIA DISPONIBLE : 3993
** AVISO: ETIQUETA 800 PRESENTADA, PERO NO UTILIZADA EN EL NIVEL 1
** AVISO: ETIQUETA 300 PRESENTADA, PERO NO UTILIZADA EN EL NIVEL 1
**** ACABO LA COMPILACION ***  0 ERRORES **
```

Handwritten annotations:

VER 200; Se genera: GOTO 4;

Se define etiqueta 300, nombre interno (5).

I := 0;

Se define etiqueta 200. Nombre interno (4).

SI NO I ENTONCES

VER 900; Se genera: GOTO 1;

FIN de condicional sencilla.

AVISOS.

```
(*$S++*)
(*$G+*)
PROGRAM LENGEST;
USES TRANSCEN, CODGRAL, INICOD;
LABEL 1, 2, 3, 4, 5, 6, 7, 8, 9, 10;
PROCEDURE INI11;
BEGIN
 INICIALI;
 NIVEL:=1;
 TOPEMEM:=DISPLAY[1]+2;
 END; (* INI11 *)
 BEGIN
 INI11;
 AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)   } I:=1;
 SELEC[0]:=0;
 MUEVE(1,2,1, 1.00000,1);
 TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
   1 : ;   2 : ;   3 : AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
 NIVELUNO:=1;                                           } SI I < LIMITE EN-
 OPERBIN(MENOR,1,2, 2.00000,1);                           TONCES...
      (* TERMINO EL PROCESAMIENTO DE LA EXPRESION ***)
 EVALCOND(-1,2);
 IF CONDRES THEN BEGIN
 AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
 NIVELUNO:=1;
 OPERBIN(SUMA,1,2, 1.00000,1);                          } I:= I+1;
 SELEC[0]:=0;
 MUEVE(1,2,1,-1,2);
 TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
 GOTO 3; <------------------------------------------ VER 500;
 END ELSE BEGIN <---------------------------- SINO I < LIMITE
 GOTO 4; <------------------------------------------ VER 200;
 END; (* SINO *) <--------------------------- Termina condicional.
   5 : AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
 SELEC[0]:=0;                                           } I:=0;
 MUEVE(1,2,1, 0.00000,1);
 TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
   4 : AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
 NIVELUNO:=1;
 OPERUNA(NEGACION,1,2);                                 } SI NO I ENTON-
      (* TERMINO EL PROCESAMIENTO DE LA EXPRESION ***)    CES...
 EVALCOND(-1,2);
 IF CONDRES THEN BEGIN
 GOTO 1; <------------------------------------------ VER 900;
 END; (* SI *) <----------------------------- Termina Condicional.
 CLOSE(LST,LOCK);
 END.
```

Figura 6.33 Programa Objeto Correspondiente al Ejemplo de la Figura 6.31

```
RESOPNDO   1.00000 1 0 1 1
COMOPNDO   1.00000
RESOPNDO   1.00000 2 1 2 1
********* MUEVE *** OPNDO1 2 1 OPNDO2 1 0VALOR   1.00000
RESOPNDO   2.00000 1 0 1 1
COMOPNDO   2.00000
RESOPNDO   1.00000 2 1 2 1
COMOPNDO   1.00000
OP. BIN. 2 OPNDO1 2 1  1.00000 OPNDO2 1 0  2.00000 RES 2 2  1.00000
METE DES. IND 0 TIPO 2 LOC. 2
LONG.:  TIPO 2 LOC. 2 LONGITUD 1
RESOPNDO -1.00000 2 SACA DES. IND 0 TIPO 2 LOC. 2
2 2 1
COMOPNDO   1.00000
RESOPNDO   1.00000 1 0 1 1
COMOPNDO   1.00000
RESOPNDO   1.00000 2 1 2 1
COMOPNDO   1.00000
OP. BIN. 6 OPNDO1 2 1  1.00000 OPNDO2 1 0  1.00000 RES 2 2  2.00000
METE DES. IND 0 TIPO 2 LOC. 2
LONG.:  TIPO 2 LOC. 2 LONGITUD 1
RESOPNDO -1.00000 2 SACA DES. IND 0 TIPO 2 LOC. 2
2 2 1
COMOPNDO   2.00000
RESOPNDO   1.00000 2 1 2 1
********* MUEVE *** OPNDO1 2 1 OPNDO2 2 2VALOR   2.00000
RESOPNDO   2.00000 1 0 1 1
COMOPNDO   2.00000
RESOPNDO   1.00000 2 1 2 1
COMOPNDO   2.00000
OP. BIN. 2 OPNDO1 2 1  2.00000 OPNDO2 1 0  2.00000 RES 2 2  0.00000
METE DES. IND 0 TIPO 2 LOC. 2
LONG.:  TIPO 2 LOC. 2 LONGITUD 1
RESOPNDO -1.00000 2 SACA DES. IND 0 TIPO 2 LOC. 2
2 2 1
COMOPNDO   0.00000
RESOPNDO   1.00000 2 1 2 1
COMOPNDO   2.00000
OP. UNA. 15 OPNDO1 2 1  2.00000 RES 2 2  0.00000
METE DES. IND 0 TIPO 2 LOC. 2
LONG.:  TIPO 2 LOC. 2 LONGITUD 1
RESOPNDO -1.00000 2 SACA DES. IND 0 TIPO 2 LOC. 2
2 2 1
COMOPNDO   0.00000
```

I := 1;

I < LIMITE. I vale 1.0 ⇒ el resultado es VERDADERO (1).

Se toma resultado de la comparación anterior.

I := I+1; I vale 2.0

I < LIMITE I vale 2.0. LIMITE=2

El resultado de la comparación es FALSO (0)

Se evalúa NO I. Resulta FALSO.

La Condicional no es ejecutada.

Figura 6.34 Ejecución del Programa Objeto de la Figura 6.33

I) RESUMEN DE MODIFICACIONES AL ANALIZADOR SEMANTICO

En la versión anterior del compilador, el analizador semántico solamente tenía implantado la Tabla de Símbolos y la declaración de variables. Además se diseñaron una serie de estructuras y procedimientos para instaurar otros elementos del lenguaje. De esa versión se tomaron algunas ideas en su forma original y otras más se modificaron. Varios de esos cambios ya se han discutido en el transcurso de este capítulo y la razón de esta sección es presentar un breve resumen de todas las modificaciones efectuadas, indicando la sección dentro de la tesis, en donde son comentadas más ampliamente.

i) El cambio más importante que se dió al analizador semántico y al compilador es que el reconocimiento ahora se dirige por "llamados semánticos" y no por un seguimiento del árbol sintáctico, nodo por nodo, como ocurría anteriormente.
En las secciones V.C.2 y VI.B se habla más detalladamente de los "llamados semánticos".

ii) Anteriormente, el descriptor de una constante real constaba de un descriptor primario y de un descriptor adicional. En el campo 2 de este descriptor adicional se colocaba el valor de la constante.

Este diseño es erróneo porque los campos de los descriptores son enteros y la constante es real y además desperdicia espacio. Actualmente sólo se emplea al descriptor primario y en el campo 4 hay un apuntador al diccionario de átomos, a donde está definida la constante.

La rutina VALOR se encarga de procesar la definición de la constante y de entregar su valor decimal. Ver las secciones VI.C.2 y VI.D.2 .

iii) Los descriptores de las estructuras Gráfica, Tabla e Histograma anteriormente contenían en el campo 4 (localidad en memoria de la variable) un cero. En la nueva versión se

almacena la dirección en memoria del primer campo de la estructura, es decir la dirección de la estructura. De esta forma se obtiene mayor comodidad en el manejo de estas variables, como ocurre en la generación a código del llamado a la rutina ASIGEST, que asigna una estructura a otra. Ver las secciones VI.C.2 y VI.F.

iv) En la estructura Gráfica, el número de descriptores adicionales era seis, ahora son siete. Ello se debe a que antes en un sólo descriptor se colocaban los máximos y mínimos de las abscisas y ordenadas, quebrantando la regla de que el último campo se usa únicamente para indicar el número de descriptores adicionales a aquel.

Ahora el máximo y mínimo de las abscisas se guardan en un descriptor y los de las ordenadas en otro. Ver la sección VI.C.2 .

v) En la versión en FORTRAN se pensaba almacenar una cadena en 30 celdas de memoria. En la versión actual una cadena solo ocupa 16 celdas y además una de ellas indica la longitud real de la cadena. Ver la sección VI.D.2 .

vi) En la versión actual los arreglos deben declararse con el límite inferior igual a 1 en cada dimensión y el tamaño de cada una de ellas debe ser de al menos de dos elementos.

Además, la operación de dimensionamiento ahora se efectúa hasta finalizar cada operación que involucre uno o dos arreglos. Ver la sección VI.D.2 .

vii) En la versión anterior se tenía pensado emplear un stack de operadores para la compilación de expresiones. En esta nueva versión ello no fué necesario. Ver la sección VI.E.2.

viii) Se idearon en la versión anterior una serie de rutinas para realizar en ejecución la evaluación de una expresión, ellas son:

OPERACIONBINARIA (LOC1, TIPO1, LOC2, TIPO2, LOC3, TIPO3)
OPERACIONUNARIA (LOC1, TIPO1, LOC3, TIPO3)

donde OPERACIONBINARIA y OPERACIONUNARIA son los nombres de las rutinas para cada una de las operaciones en el lenguaje (suma, resta, negación, determinante, etc...). LOC1, TIPO1, LOC2 y TIPO2 contienen la localidad y tipo del primer y segundo operandos de la operación, mientras que LOC3 y TIPO3 contienen la localidad y tipo del operando resultado. Estos dos últimos parámetros son determinados hasta ejecución.

Los tipos de operando que se habían definido son: Real, Arreglo Real, Texto y Arreglo Texto. En la versión actual se incluyeron adicionalmente los tipos: Literal Real, Gráfica, Tabla, Histograma y Cuerpo de Gráfica. (Ver la sección VI.E.3).

En la versión actual se decidió tener una sola rutina para todas las operaciones binarias y otra rutina para todas las unarias. Ellas son:

OPERBIN (OPER, LOC1, TIPO1, LOC2, TIPO2) y
OPERUNA (OPER, LOC1, TIPO1)

donde OPER es el nombre de la operación y los otros parámetros representan la localidad y tipo del primer y segundo operandos. La descripción del operando resultado se deja en dos variables globales: LOC3 y TIPO3.

También se había ideado el stack de descriptores de operandos que almacenara la localidad y el tipo de los resultados temporales que surgen al evaluar una expresión. (Ver la sección VI.E.5). Dicho stack se implantó en la versión actual y cumple un importante papel en la evaluación de expresiones.

Para administrar el stack anterior se idearon un par de rutinas que fueron instauradas en la versión actual y ellas son:

METE (LOC, TIPO) y SACA (LOC, TIPO).

También se tomó la idea de tener una rutina que entregara la longitud de un operando dado, a partir de su localidad y tipo, con el fin de actualizar al tope del stack conforme se evalúa una expresión. Esa rutina actualmente luce como:

LONG (LONGITUD, LOC, TIPO).

Para asignar una estructura a otra y poder implantar la proposición de asignación se ideó la rutina:

MUEVE (LOC1, TIPO1, SELEC1, LOC2, TIPO2, SELEC2)

donde LOC1 y TIPO1 describen al operando de donde se tomará la información. SELEC1 es un vector que indica los elementos que se tomarán en caso de que el operando represente a un arreglo. Los otros tres operandos representan lo correspondiente para el operando que recibirá la información.

Esta rutina se implantó como:

MUEVE (LOC1, TIPO1, NIVEL1, LOC2, TIPO2)

donde LOC1, TIPO1 y NIVEL1 son generados en compilación y representan el desplazamiento, tipo y nivel de bloque a que pertenece la variable que recibirá la información y LOC2 y TIPO2 describen al operando de donde se tomará la información. En vez de manejar a los vectores SELEC1 y SELEC2 como parámetros, se toma un sólo vector: SELEC, que es una variable global y se utiliza para indicar a que elementos se asignará la estructura descrita por LOC2 y TIPO2.

Se determinó manejar un sólo vector, porque es difícil detectar al momento de compilar una asignación, si la expresión a asignar se encuentra afectada por una expresión selectiva de arreglo. En este caso la expresión selectiva es compilada como parte de la expresión a asignar.

Para completar la implantación de expresiones y asignaciones se diseñaron varias estructuras y rutinas que no se habían contemplado antes, como en el caso de la implantación de arreglos explícitos.

El código que actualmente se genera es más compacto que el que se pensaba generar, pues muchos llamados a rutinas que se pensaban poner en el programa objeto están contenidos dentro de otros llamados, como ocurre con los llamados a METE y a SACA que son hechos dentro de OPERBIN y OPERUNA.

Una revisión detallada de la implantación de expresiones y de la proposición de asignación y de las rutinas y estructuras de datos involucradas, se encuentra en las secciones E, F, J y K de este capítulo.

ix) La expresión selectiva de arreglo se había diseñado con las características de que ningún elemento se pudiera repetir dentro de una misma dimensión y además, que el orden en que se seleccionaran los elementos sería el dado por su posición dentro del arreglo. La expresión se implantó permitiendo la repetición de elementos y estableciendo que el orden en que se seleccionen los elementos es según aparezcan en la expresión. Ver la sección VI.E.7 .

x) En la definición gramatical de la proposición de asignación se corrigió un error consistente en que del lado izquierdo de una asignación, no podía haber un identificador afectado simultáneamente por un campo y por una expresión selectiva de arreglo. Ver las secciones IV.D y VI.F.

xi) El diseño que se había ideado para implantar la proposición condicional fué modificado. Se basaba en la administración de un stack de etiquetas y en la generación de líneas de brinco incondicional ("GOTO").

La implantación actual se basa en el manejo de un stack de proposiciones y genera líneas de proposiciones condicionales de PASCAL. Ver la sección VI.G.

Las razones por las que se decidió cambiar el diseño se pueden resumir en:

i) Se genera código más limpio.

ii) Es más fácil generar el código como está actualmente, que estar emitiendo etiquetas distintas por cada proposición condicional.

iii) Se requieren los mismos llamados semánticos para hacer validaciones y generar código en ambos diseños.

iv) En el stack de etiquetas se requiere almacenar marcas para identificar proposiciones condicionales compuestas*, en el otro stack no.

v) El stack de proposiciones puede ser empleado para procesar otras proposiciones de control, el de etiquetas no, pues no hay forma de saber a que tipo de proposición corresponde cada etiqueta.

---

* Es decir, se requiere poner una marca especial que represente al "INICIO".

J) PRINCIPALES ESTRUCTURAS DE DATOS Y RUTINAS DEL ANALIZADOR SEMANTICO

Estructuras de Datos

Los nombres de archivos y arreglos aparecen en mayúsculas, los de escalares y constantes en minúsculas.

Archivos:

Nombre Lógico	Nombre Físico	Función
LST	Los proporciona	Listado de semántica
COD	el usuario	Programa objeto
RESUNO	SEMCHARS	Diccionario de identificadores, números
RESDOS	SEMVDT	y cadenas
ANSIN	ANREC	Arboles sintácticos

Constantes:

maxnivel.- Máximo nivel al que pueden anidarse bloques. (10).

maxdim.- Máximo número de dimensiones que puede tener un arreglo. (20). (Cada dimensión cuenta por 2).

maxparam.- Máximo número de parámetros en una rutina (50).

maxchar.- Longitud máxima de una cadena (60 caracteres).

maxet.- Máximo número de etiquetas por bloque (10).

Variables:

faults.- Contador de errores en análisis semántico.

nlínea.- Número de línea procesada. La que contiene el listado del análisis lexico-sintáctico.

nivel.- Nivel lexicográfico en proceso.

minimamemoria.- Para estimar la mínima memoria disponible durante la compilación.

Tabla de Símbolos:

CHARS y VARDIC.- Diccionario de átomos.

DES.- Descriptores.

LIG, LI, INFO.- Estructuras adicionales.

NIVELS.- Arreglo de niveles, apunta a DES.

icht, ivdt, apdes, apdis.- Apuntadores a la Tabla de Símbolos.

Arbol Sintáctico:

A.- Arreglo de alternativas.

NP.- Arreglo de apuntadores o nodos hermanos.

ap.- Apuntador al árbol.

op.- Valor de la alternativa del nodo ap. (A ap).

STOPNDO.- Stack de operandos.

STOPDOR.- Stack de operadores.

iopndo,iopdor.- Apuntadores a STOPNDO y STOPDOR.

OPER.- Nombres de las operaciones binarias y unarias a generar al código.

TIPORES.- Arreglo para obtener el tipo de resultado de una operación binaria.

ARREXP.- Para almacenar la información referente a un arreglo explícito.

esap.- Número de elementos asignados al arreglo SELEC, que se han generado al programa objeto. Para expresiones selectivas de arreglo.

TGOTO.- Diccionario de etiquetas.

indgt.- Apuntador a TGOTO.

STPROP.- Stack de proposiciones.

indstprop.- Apuntador a STPROP.

PROPOP.- Arreglo de banderas para revisar que no haya declaraciones después de proposiciones en cada nivel.

monitor.- Variable para emitir o no la reseña del análisis semántico.

Rutinas:

El analizador semántico consiste de un gran número de rutinas, algunas de ellas son rutinas de biblioteca agrupadas en unidades de la biblioteca de U.C.S.D. PASCAL.

El analizador semántico está estructurado en tres unidades y el programa principal. Las unidades son:

GLOBAL.- Contiene variables y rutinas utilizadas en todo momento de la compilación.

INIYFIN.- Contiene las rutinas que inician y finalizan a cada nivel o bloque y también al análisis semántico.

DECLARA.- Contiene las rutinas empleadas para declarar variables y estructuras del lenguaje estadístico.

Veamos las rutinas principales de cada unidad y del programa principal.

UNIDAD GLOBAL:

GEN,GENPARS,GENPARR,GENPARE.- Rutinas encargadas de generar el código.

MINMEM.- Para estimar la mínima memoria disponible.

ERROR.- Para emitir y contabilizar errores semánticos.

TEST.- Valida que el valor de una variable real no exceda la capacidad de una variable entera de U.C.S.D. PASCAL.

VEDESC, VESIG.- Para ver el contenido de descriptores de la Tabla de Símbolos.

ENCNIV y GENNIV.- Determinan el nivel en que fué declarada una variable y si es necesario generar al código dicha información.

VALOR.- Decodifica y entrega el valor en decimal de una cadena numérica, del diccionario de átomos.

NUMOCONS.- Determina el valor de un número o de una constante numérica.

LEEAN.- Lee un árbol sintáctico.

MSTPROP,VSTPROP,SSTPROP.- Para revisar y administrar el stack de proposiciones.

UNIDAD INIYFIN:

ININIVEL.- Inicializa estructuras y apuntadores al comenzar un bloque.

INICIA.- Inicializa a estructuras, apuntadores y archivos al comenzar el analizador semántico.

FINNIVEL.- Valida estructuras y finaliza el análisis de un bloque.

FINSEMANTICA.- Valida estructuras, cierra archivos y da por terminado el análisis semántico.

UNIDAD DECLARA:

PONNIV y DESCRI.- Insertan un descriptor a la Tabla de Símbolos.

DECCTE.- Declara una constante numérica o tipo texto.

DECESC.- Declara una lista de variables escalares tipo texto o tipo real.

DECARREGLOS.- Declara una lista de arreglos.

DECEST.- Declara una lista de gráficas o una lista de tablas.

DECHIST.- Declara una lista de histogramas.

PROGRAMA PRINCIPAL:

El programa principal, dada su magnitud está distribuido en cuatro archivos. Dos archivos contienen el código para procesar expresiones y se llaman EXPR.UNO y EXPR.DOS. El tercer archivo se llama COTOYCOND y contiene las rutinas para procesar la proposición condicional y la de transferencia. El cuarto archivo contiene al programa principal y llama en compilación a los otros tres. El programa principal se llama SEMANTICA8.

Para posteriores modificaciones al analizador semántico se deberá cambiar el número que trae el nombre del mismo, por cada nueva versión que se vaya haciendo.

Pasemos a ver las rutinas que componen al programa principal:

MOPNDO, SOPNDO.- Para manejar el stack de operandos.

MOPDOR, SOPDOR.- Para manejar el stack de operadores.

EXPSELEC.- Procesa una expresión selectiva de arreglo.

IDCONCAMPO.- Procesa un identificador que está referenciado junto con un campo de estructura.

PROCEXP.- Procesa a una expresión.

Dentro de PROCEXP se encuentran las siguientes rutinas:

INIARREXP, ARREXPALM, ARREXPNUM, ARREXPFNUM, ARREXPEXP, ARREXPFIN.- Analizan un arreglo explícito.

METEOPNDO.- Analiza un operando y lo inserta al stack de operandos si es correcto.

OPBINARIA.- Procesa una operación binaria.

OPUNARIA.- Procesa una operación unaria.

VALIDAOPNDO.- Valida que un escalar no sea usado como arreglo.

Las rutinas EXPSELEC, IDCONCAMPO y las de manejo de los stacks de operandos y operadores están fuera de PROCEXP, porque pueden llamarse al analizar el lado izquierdo de una asignación,

el que se procesa mediante la rutina ASIGNACION.

ASIGNACION.- Procesa una proposición de asignación. Se ayuda mucho de PROCEXP, de EXPSELEC y de IDCONCAMPO.

BUSCRET,DECET,BRINCA.- Procesan la aparición de etiquetas y la proposición de transferencia incondicional.

CONDICIONAL,ALTCONDICIONAL.- Procesan una proposición condicional.

SECCONDICIONAL.- Procesa la instrucción de "FIN" que puede corresponder al final de una proposición compuesta, de una rutina (en el futuro) o del programa fuente.

TERCONDICIONAL.- Concluye todas las proposiciones sencillas que estén anidadas, al llegar a un punto y como o a un "FIN".

SEMANT.- Procesa una frase sintáctica del programa fuente a través de su árbol sintáctico. Esta rutina se encarga de llamar a todas las demás rutinas del analizador semántico, de acuerdo a los llamados que vaya encontrando en el árbol.

K) PRINCIPALES ESTRUCTURAS DE DATOS Y RUTINAS DEL PROGRAMA OBJETO

Como ya se debe tener muy en claro, el programa objeto consiste de un programa escrito en PASCAL. Este programa se compone principalmente de llamados a rutinas de biblioteca y de referencias a variables que se definen en esas mismas rutinas.

Dichas rutinas se encuentran distribuidas en tres unidades de la biblioteca del sistema U.C.S.D. PASCAL. Sin embargo, el programa objeto también invoca a rutinas definidas en el mismo, que se utilizan para inicializar el espacio dispuesto a arreglos en cada bloque o nivel lexicográfico y en el futuro se utilizarán para traducir las rutinas declaradas en el programa fuente.

Para el actual momento de implantación del compilador, sólo se genera la rutina INI11, que se emplea para inicializar al programa objeto y a los arreglos y estructuras que son globales en el programa fuente.

Las rutinas que se generen al programa objeto se deben componer del prefijo "INI" seguido del nivel lexicográfico a que corresponde y del número de rutina dentro de ese bloque.

Pasemos ahora a ver las principales constantes, variables y rutinas de la biblioteca del programa objeto. Como en ocasiones anteriores, las constantes y variables escalares se escriben en minúsculas y las demás estructuras en mayúsculas.

Estructuras de Datos:

Archivos:

Nombre Lógico	Nombre Físico	Función
LST	Lo proporciona el usuario	Proporciona una reseña de todas las acciones que hace el programa objeto.
FILECH	SEMCHARS	Diccionario de cadenas que emplea el programa objeto.

Constantes:

minmem, tanmem.-	Límites para el stack de memoria. (0 y 2000 respectivamente).
maxnivel.-	Máximo nivel de anidamiento de rutinas. Es el mismo de compilación (10).
maxdim.-	Número máximo de dimensiones en arreglos (10).
maxchar.-	Máximo número de caracteres por variable tipo texto (60).
tamcad.-	Número de elementos del stack de memoria empleados por una variable tipo texto (16).

Variables:

MEM.-	Stack de memoria.
CHARS.-	Arreglo para almacenar el diccionario de átomos SEMCHARS. Se emplea para tomar las cadenas aparecidas en el programa fuente y que usará el programa objeto.
stp.-	Variable booleana que se prende cuando los operandos corresponden a variables tipo texto.
nivel.-	Nivel lexicográfico en que vive la ejecución del código.

topemem.- Tope al stack de memoria MEM.

auxtop1.- Variable auxiliar para administrar el espacio en memoria dedicado a temporales.

DISPLAY.- Arreglo de registros de despliegue.

STOPNDO.- Stack de descriptores de operandos que son resultados temporales.

istopndo.- Tope al stack STOPNDO.

ARREXP.- Arreglo para construir arreglos explícitos.

indarrexp.- Variable auxiliar empleada para almacenar las dimensiones de un arreglo explícito.

SELEC,SELECB,LOCSEL,LOCRES.- Estructuras para llevar a cabo una expresión selectiva de arreglo.

locarr.- Variable auxiliar para almacenar las dimensiones a una variable tipo arreglo.

condres.- Variable booleana que contiene la interpretación lógica de la expresión de una proposición condicional.

Rutinas:

Como se acaba de mencionar, la mayoría de las rutinas que invoca el programa objeto residen en tres unidades de biblioteca. Veamos dichas unidades:

TRANSCEN.- Unidad intrínseca del sistema PASCAL. Se tiene que cargar al código para poder utilizar la función de logaritmo, con la que se efectúa la operación de potencia.

CODGRAL.- Contiene todas las rutinas para ejecutar expresiones, asignaciones, condicionales y manejo de etiquetas.

INICOD.- Inicializa al programa objeto.

Veamos ahora de que consisten las dos últimas unidades.

UNIDAD INICOD:

Contiene una sola rutina, llamada INICIALI que se encarga de abrir el archivo para el listado de la ejecución, carga el diccionario de cadenas, inicializa en ceros al stack de memoria y también inicializa a otras variables.

UNIDAD CODGRAL:

ERROR.- Se encarga de emitir al listado, el error encontrado.

METE, SACA.- Para manejar el stack de operandos temporales STOPNDO.

TEST.- Verifica que un número real no exceda la capacidad de una variable entera de U.C.S.D. PASCAL.

RESOPNDO,COMOPNDO.- Obtienen a partir del operando generado en compilación (LC,TP), el verdadero contenido del operando en ejecución (LOC, TIPO). Además, si el operando corresponde a una variable escalar, se prende una bandera (ETP1 ó ETP2) y se deja el valor de ella en una variable global (VAL1, VAL2 ó VAL3).

LONG.- Determina la longitud de un temporal, en celdas de memoria y la deja en la variable global ilong.

DIRECCONS,DIRECUAR.- Rutinas para obtener la posición de un elemento de un arreglo en memoria.

ESCCAD, ESCARREGLO.- Emiten al listado del programa objeto el contenido de un escalar tipo texto o de cualquier arreglo, respectivamente.

LECCAD, LEC2CAD, ASICAD.- Para cargar una cadena a una variable en memoria, al tope del stack de memoria o para asignarla a otra variable, respectivamente.

DEC1ARR, DEC2ARR.- Para almacenar el número de dimensiones y el tamaño de cada una de ellas a una variable tipo arreglo. Se invocan dentro de la rutina de inicialización INI11.

OPERDIN.- Efectúa una operación binaria.

OPERUNA.- Lleva a cabo una operación unaria.

AXARREXP, BXARREXP, CXARREXP, DXARREXP, CONSARREXP.- Para construir la estructura ARREXP y luego construir un arreglo explícito.

DECODSELEC.- Decodifica la información en el arreglo SELEC, para una expresión selectiva de arreglo y construye las estructuras SELECB, LOCSEL y LOCRES.

RUTSEL.- A partir de la información que DECODSELEC deja en SELECB, LOCSEL y LOCRES, obtiene un subarreglo o asigna una expresión a un subarreglo.

MUEVE.- Asigna una expresión a una variable. En caso de asignar a un subarreglo se apoya en DECODSELEC y RUTSEL.

CONSELARR.- A partir de un arreglo, aplica la expresión selectiva de arreglo y obtiene un subarreglo. Se basa en DECODSELEC y en RUTSEL.

GRABAES, LEEES.- Graba y lee el contenido de la estructura SELEC en memoria respectivamente. Se emplean para lograr la recursividad en la expresión selectiva de arreglo.

ASIGEST.- Asigna una variable de tipo estructura (Gráfica, Tabla o Histograma) a otra variable del mismo tipo.

EVALCOND.- Interpreta una expresión como expresión lógica o booleana y entrega el resultado en la variable lógica CONDRES.

La unidad INICOD se encuentra escrita en el archivo INICIALI.TEXT, mientras que la unidad CODGRAL se encuentra repartida en tres archivos, que son: UCODUNO.TEXT, UCODDOS.TEXT y UCODTRES.TEXT.

VII. MODIFICACIONES Y MEJORAS PROPUESTAS AL COMPILADOR

La implantación actual del compilador todavía no contempla todas las características del lenguaje estadístico, sin embargo basta con agregar algunos elementos para tener un producto que ya pueda ser utilizado por los usuarios y que contenga las ventajas del lenguaje estadístico, consistentes en el manejo sencillo y eficiente de datos a través de matrices y en la emisión de reportes en forma de gráficas, histogramas y tablas.

Esos elementos faltantes se detallan a continuación.

i) No se han incluido en el programa objeto los algoritmos para obtener la inversa y la determinante de una matriz. Esto se debe a que la implantación de estas dos operaciones es poco importante dentro de todo el compilador y porque la implantación de ellas en el código consiste básicamente en incluir los algoritmos, que pueden encontrarse en cualquier libro de análisis numérico y representan un buen ejercicio para comenzar la siguiente parte del desarrollo del compilador.

ii) Falta implantar el cuerpo de una gráfica y la operación de suma o unión entre gráficas. La importancia de esta estruc

tura podrá constatarse sólo hasta tener concebida la salida de reportes tipo gráfica.

El cuerpo de una gráfica originalmente fué diseñado como una matriz que albergue la imagen de la gráfica, es decir los puntos a graficar. Como cada elemento del cuerpo es un caracter, entonces se puede concebir al cuerpo como un arreglo bidimensional de caracteres, almacenando cuatro caracteres por cada celda de memoria. Este diseño aunque no desperdicia espacio es costoso. Si por ejemplo, se tiene una gráfica de 50 renglones por 100 columnas, se requieren de 5000 caracteres para almacenar la imagen de la gráfica, lo que equivale a 1250 celdas de memoria.

Para ahorrar espacio en memoria se puede diseñar un tipo especial para el cuerpo de la gráfica, de tal forma que cada elemento ocupe menos de ocho bits o también se puede manejar el cuerpo en la memoria secundaria (como el disco), sobre archivos temporales que no ocupen espacio en el stack de memoria.

iii) Falta implantar toda la estructura de entrada-salida del código, que consiste en el manejo de archivos, de formatos, en los procedimientos de lectura y escritura de datos y en la emisión de tablas, gráficas e histogramas.

Teniendo estos tres puntos implantados en el compilador, el producto ya puede ser realmente empleado por cualquier usuario, pues faltarían por implantar estructuras de control más poderosas y también flatarían las rutinas y funciones, que aunque son elementos muy importantes dentro del lenguaje, se puede prescindir de ellos en programas pequeños y no muy complicados.

Las mejoras que se proponen al compilador para obtener un producto más eficiente se detallan en las siguientes líneas.

iv) Inicialización de Variables.

Actualmente se inicializa en ceros a todas las celdas del stack de memoria al momento de comenzar la ejecución del programa objeto. La inicialización cubre a todo el stack y no únicamente

al espacio correspondiente a las variables del programa.

La inicialización se lleva en esta forma debido a que durante la compilación, sólo hasta después de haber procesado todas las declaraciones, se puede determinar el espacio que requerirá el bloque para almacenar sus variables, es decir, sólo hasta después de haber emitido las rutinas que incializarán los descriptores dinámicos de los arreglos en el programa objeto, se puede determinar el número de celdas a inicializar.

Se puede idear un mecanismo que optimice el procedimiento de inicialización, como podría ser, que al determinar el espacio requerido, se modifique al programa objeto, en el llamado a la rutina que se encarga de hacer la inicialización.

v) Manejo de Temporales.

No hay una reutilización del espacio dedicado a almacenar operandos temporales durante la evaluación de expresiones debido a que los operandos son de longitud variable y ésta, únicamente puede determinarse en la ejecución. Sin embargo puede resultar necesario disponer de un manejador de temporales, en casos de expresiones que manejen un gran volumen de datos y para ello se requiere estimar en ejecución, el espacio que ocupará cada temporal a obtener.

Aunque es complicado, el manejador volverá más eficiente la administración del espacio libre en el stack de memoria.

vi) Diccionario de Números.

Actualmente los números se manejan como cadenas de caracteres que viven en el diccionario de átomos junto con los textos y los identificadores.

A fin de ahorrar espacio en el arreglo de caracteres y mejorar la eficiencia en el manejo de números se podría disponer de un arreglo de reales para almacenar a los números. Este arreglo formaría parte del diccionario de átomos que quedaría constituido por tres arreglos: VARDIC, CHARS y NUMEROS.

Los elementos de VARDIC que fueron positivos apuntarían a

CHARS, mientras que los negativos lo harían a NUMEROS, en donde viven números reales.

Este cambio implicaría modificar tanto al reconocedor lexicográfico como al semántico.

vii) Mensajes de Error.

En la implantación original, los errores de compilación se emitian mediante un pequeño mensaje. Actualmente sólo se emite un número para identificar al error, ello se debe a que antes, la rutina de error contenía todos los mensajes de error y por medio de un parámetro se emitía el mensaje correspondiente. Debido a lo costoso e ineficiente que resultaba ese manejo de errores se decidió emitir solamente el número de error.

Para volver a emitir mensajes de error se puede diseñar un archivo que los contenga y cuya presencia sea opcional. Si el compilador encuentra dicho archivo, al encontrar errores buscará el mensaje correspondiente y lo emitirá al listado de compilación. Si el archivo no está presente, sólo se emitirá el número de error.

El archivo puede consistir de registros, cada uno de ellos consistente de caracteres y de tamaño fijo; por ejemplo, registros de 80 caracteres.

viii) Hash en la Tabla de Símbolos.

El método que actualmente emplea la Tabla de Símbolos para buscar e insertar elementos es lineal, el cual se va haciendo más tardado conforme aumenta el tamaño de la Tabla. Se podría implantar un método de Hash que volviera eficiente la búsqueda e inserción de elementos.

ix) Compilador en un sólo Programa.

Se podría intentar unir a los analizadores lexicográfico y semántico en un sólo programa, cuando el reconocedor semántico ya esté concluido.

El compilador tendría que modificarse para que cada vez que se termine de construir un árbol sintáctico, se llame al reconoce-

dor semántico inmediatamente y se analice así, a cada línea en un sólo paso.

Entre las ventajas que se obtendrían, se evitaría tener que dejar a los árboles sintácticos y al diccionario de átomos en archivos temporales.

x) Diccionario de Textos en el Programa Objeto.

Como se mencionó en su momento, si el programa objeto hace referencia a textos aparecidos en el programa fuente, se requiere que el arreglo de caracteres del diccionario de átomos esté presente durante la ejecución del programa objeto, a fin de poder disponer de los textos. Esto se decidió para disminuir el número de líneas emitidas al código.

Se puede diseñar una alternativa a fin de que se simplifique el manejo de los textos en el programa objeto.

xi) Magnitud del Programa Objeto.

Al compilar programas objeto en donde el número de líneas de asignaciones y llamados a rutinas que hay en el programa principal es mayor a 90, ocurren problemas, pues se excede la capacidad del compilador (U.C.S.D. PASCAL para APPLE, versión 1.0) para procesar un bloque.

Para dar solución al problema en la actual implantación del compilador, se divide al programa objeto en bloques. A cada bloque se le asigna a lo más, 90 líneas, pues se ha visto que bloques de esta magnitud son soportados por el compilador de PASCAL sin problemas.

Esta modificación del programa objeto por parte del usuario funciona, siempre y cuando cada una de las proposiciones condicionales y cada una de las proposiciones de transferencia incondicional pueden resolverse perfectamente en un sólo bloque. Si por ejemplo, la etiqueta α aparece en la línea X y la transferencia a α se encuentra en la línea X+200, la solución al programa objeto se vuelve complicada y es mejor modificar el programa fuente.

La implantación de rutinas en el compilador estadístico puede ser una solución al problema, pero está latente la desventaja de que las rutinas en el programa fuente deban ser pequeñas.

También se puede tratar de aumentar aún más al "Run-time support", como en el caso de las asignaciones a los arreglos auxiliares ARREXP y SELEC (utilizados para generar arreglos explícitos y para aplicar la expresión selectiva de arreglo), que podrían simplificarse.

Otra alternativa consistiría en transportar al compilador a otra máquina, como la computadora PC ya sea al mismo U.C.S.D. PASCAL para PC, (con él tiene completa compatibilidad) o a otro compilador, como el TURBO-PASCAL, (aunque hay diferencias entre U.C.S.D. y TURBO, las modificaciones requeridas son pocas, la mayoría en las instrucciones de acceso a archivos). La gran ventaja que tiene la PC y el software de la PC sobre la APPLE II, en cuanto a recursos en memoria y en discos duro y flexible, representa una gran ganancia en capacidad para mejorar al compilador estadístico y para procesar programas fuentes más grandes.

xii) Una alternativa fácil de tomar, para que el compilador pueda ya ser empleado, sin tener implantadas las proposiciones de entrada y salida, consiste en comentar al programa objeto indicando las celdas en el stack de memoria que corresponden a cada variable declarada. De esta forma el usuario podría modificar el programa objeto agregando instrucciones de lectura y escritura propias a los requerimientos del usuario y en general, cualquier conjunto de instrucciones que el usuario desee incluir.

Por ejemplo, si el programa fuente contiene: ARREGLO A[1:10], el programa objeto contendría: DEC1ARR(LOC,1,10); (* Arreglo A, cuyas celdas viven entre DISPLAY[NIVEL] + LOC + 2 y DISPLAY[NIVEL] + LOC + 11 *).

El usuario podría agregar:

```
FOR UI: = DISPLAY[NIVEL] + LOC + 2  TO  DISPLAY[NIVEL]
    + LOC + 11 DO
          READ(MEM[UI].R);
```

donde UI sería una variable que el usuario tendría que declarar, a fin de no entrar en conflicto con las variables utilizadas por las rutinas de biblioteca del programa objeto.

VIII CONCLUSION

Primeramente quisiera resaltar que esta tesis no representa un trabajo aislado, sino que forma parte de un proyecto que comenzó con la definición del lenguaje estadístico en la tesis anterior a ésta y continuará en por lo menos una tesis más, hasta lograr una implantación completa del compilador.

Además la realización del proyecto tiene como fines primordiales, ser un apoyo computacional para los profesores, investigadores y estudiantes de estadística y ser una herramienta que sirva tanto para la enseñanza como para la investigación en el diseño de compiladores.

Las desventajas que presentaba el desarrollar el proyecto en la computadora NOVA, consistentes en la gran falibilidad de la máquina frente a cambios en la temperatura, las continuas fallas del disco duro y los pocos recursos con que contaba en equipo y en software, hicieron que al tener disponibles las computadoras FRANKLIN en los laboratorios de profesores y estudiantes de la Facultad de Ciencias, se tomará la decisión de mudar todo el compilador a las nuevas máquinas.

El trabajo que acarreó cambiar el compilador de computadora, estribó en reescribir todos los programas que componían al compilador y que estaban escritos en BASIC y FORTRAN, traduciéndolos a PASCAL y también se volvieron a reescribir los demás archivos que formaban parte del compilador, como la gramática.

La conversión del compilador permitió un desarrollo más fácil y rápido del compilador, así como mejorar la implantación anterior.

Además el compilador ahora es más portable, pues programas de PASCAL en FRANKLIN pueden transportarse a computadoras PC, sin muchas complicaciones, ni muchos cambios. Inclusive el U.C.S.D. PASCAL de PC es totalmente compatible con el U.C.S.D. PASCAL de APPLE.

El compilador aún no se ha terminado, pero basta con agregar algunos elementos (la entrada-salida), para tener un producto que ya pueda ser empleado por la gente de estadística.

La principal ventaja del código es que consiste en un programa en PASCAL, el cual con más comentarios colocados adecuadamente y lo suficientemente explicativos, permitirán al usuario que posea cierto conocimiento de la estructura del programa objeto, poder modificar el código, introduciendo sus propios procedimientos de lectura y escritura de datos y en general, poder adaptar al código a sus propios requerimientos.

Se trató de dar un diseño estructurado, con comentarios pertinentes a los programas que componen al compilador y también se trató de describirlos detalladamente en este trabajo, buscando hacer fácil la comprensión y modificación de los mismos.

El compilador no es un producto óptimo, puesto que la primera meta a obtener reside en la implantación completa del lenguaje, sin que sea con los algoritmos más eficientes que puedan encontrarse. No obstante, la optimización del compilador es una etapa que no deberá abandonarse y constituye un elemento útil para trabajos de cursos avanzados de compiladores.

Finalmente, este proyecto es un intento de incursionar en el desarrollo de lenguajes de programación, tratando de disminuir la dependencia en que nos encontramos y de acrecentar los trabajos y literatura de computación hechos para gente que habla español.

AGRADECIMIENTOS

Agradezco al Dr. Luis Legarreta Garciadiego el haber dirigido la elaboración de esta tesis. Su continuo apoyo y recomendaciones permitieron que este trabajo llegara a buen fin.

Al Act. Javier García García por su ayuda al adentrarme en el conocimiento del lenguaje estadístico y de la primera versión del compilador, así como por sus aportaciones y críticas a la elaboración del manuscrito.

A la Mat. Elisa Viso Gurovich, al Dr. Mario Magidin Matluk y al Fis. Victor Mantilla Caballero por su participación en mi formación académica y en la revisión de la tesis.

Finalmente quiero agradecer a mi madre, la Sra. Luz María Avila de Esquivel por haber mecanografiado más de una vez el manuscrito.

APENDICE A.

I) Convenciones para Escribir la Gramática.

i) Los símbolos no-terminales (o variables meta-lingüísticas), se encierran entre paréntesis.
(COND), (ITER), (SELEC).

ii) Para definir un símbolo no-terminal con una producción que consista de símbolos terminales y no-terminales se utiliza: P( ). Para definir un símbolo no-terminal en un llamado semántico, en un tipo de átomo o en una palabra se emplea: B( ).

P(DECL) = (REAL)(4011)(L DE ID);
B(REAL) = 2004; : palabra clave
B(4011) = 4011; : llamado semántico
B(IDENTIF) = 1020; : identificador (átomo)

iii) El signo de igual "=" se emplea como separador entre los dos lados de una producción. Se lee como sigue: se define como.

iv) Los símbolos terminales se encierran entre apóstrofos. (Un apóstrofo se representa con dos apóstrofos seguidos).
'[', '[[', ']]', ''' (un sólo apóstrofo).

v) Al finalizar la definición de una producción, se coloca un punto y como ";".

P(COND) = (PSI)(RC1);
P(ITER) = (REPITE)(CUERPO ITER);
B(REPITE) = 2010;

vi) Las producciones que tengan en común el símbolo no-terminal que las define, se deben escribir en forma contigua, colocando los lados derechos de las producciones, uno tras otro y separándolos por medio de un símbolo de admiración "!". Por ejemplo:

P(DECL) = (CTE)(L DE CTES)!
(REAL)(4011)(L DE ID)!
(TEXTO)(4011)(L DE ID);

El símbolo (DECL) puede derivarse en tres distintas posibilidades.
Si entre las distintas posibilidades o alternativas, hay una que es un prefijo de otra, la derivación más grande debe escribirse primero. Por ejemplo:

```
(OPERADOR) = '>=' ! '<=' ! '>' ! '<';
```

vii) Al finalizar la descripción de la gramática se debe colocar un asterisco "*".

viii) Se pueden escribir comentarios en la gramática. Para ello la línea debe comenzar con un símbolo de porcentaje "%" y el resto de la línea se interpretará como comentario.

```
%*** DECLARACIONES ******
P(DECL) = (CTE)(L DE CTES)!

...
```

ix) La producción vacía se representa sin símbolo alguno.

```
P(INI o VAC) = (INICIO)! ;
P(PRODUCCION VACIA) = ;
```

x) Los rangos en las producciones tipo-B se interpretan como sigue:

Sea B(X) = Y

Y	SIGNIFICADO
1010	Atomo Numérico
1020	Atomo Tipo Identificador
1040	Atomo Texto
y>=2000	Palabra Clave
y>=4000	Llamado Semántico

II) GRAMATICA.

```
B(FIN)=2002;
B(CTE)=2003;
B(REAL)=2004;
B(TEXTO)=2005;
B(ARREGLO)=2006;
B(GRAFICA)=2028;
B(TABLA)=2029;
B(HISTOGRAMA)=2030;
B(ARCHIVO)=2031;
B(RUTINA)=2014;
B(VAL)=2023;
B(REN)=2024;
B(COL)=2025;
B(CUERPO)=2038;
B(SINO)=2008;
B(SI)=2007;
B(ENTONCES)=2009;
B(REPITE)=2010;
B(HASTA)=2026;
B(MIENTRAS)=2012;
B(ESTO)=2011;
B(INICIO)=2001;
B(LEE)=2016;
B(SUMANDO)=2017;
B(DE)=2018;
B(CON)=2019;
B(O)=2020;
B(LIBRE)=2021;
B(EN)=2022;
B(ESCRIBE)=2027;
B(TITREN)=2032;
B(NOMREN)=2033;
B(NOMCOL)=2034;
B(TITULO)=2035;
B(MARCOCOL)=2036;
B(MARCOREN)=2037;
B(VER)=2039;
B(LA)=2042;
B(FORMATO)=2013;
B(TITCOL)=2015;
B(Y)=2044;
B(NO)=2043;
B(NUMERO)=1010;
B(IDENTIF)=1020;
B(TEXT)=1040;
B(PI)=2040;
B(E)=2041;
```

```
          %********** LLAMADOS SEMANTICOS *************%
B(4010)=4010;
B(4011)=4011;
B(4012)=4012;
B(4013)=4013;
B(4014)=4014;
B(4015)=4015;
B(5001)=5001;
B(5007)=5007;
B(5010)=5010;
B(5014)=5014;
B(5015)=5015;
B(5018)=5018;
B(5019)=5019;
B(5020)=5020;
B(5021)=5021;
B(5025)=5025;
B(5026)=5026;
B(5027)=5027;
B(5028)=5028;
B(5029)=5029;
B(5030)=5030;
B(5031)=5031;
B(5032)=5032;
B(5033)=5033;
B(5034)=5034;
B(5035)=5035;
B(5036)=5036;
B(5037)=5037;
B(5038)=5038;
B(5039)=5039;
B(5040)=5040;
B(5041)=5041;
B(5042)=5042;
B(5043)=5043;
B(5044)=5044;
B(5045)=5045;
B(5046)=5046;
B(5047)=5047;
            %******** DECLARACIONES Y PROPOSICIONES *************
P(PRINCIPIO)=(INIOVAC)(P)   ; ;
P(P)= (DECL) ! (L ETIQ) (P GRAL)(5047) !   ;
P(INIOVAC) = (INICIO) !   ;
P(P GRAL) = (COND) ! (ITER) ! (SELEC) ! (P INC);
P(RFI) = (HAS) ! (RC1);
P(L ETIQ) = (5042)(NUMERO)   :   (L ETIQ) ! ;
P(P INC) = (PR INC)(TERMINA);
P(TERMINA)=(FIN)(5045)(RFI)(TERMINA) ! ;
P(PR INC) = (ASIG) ! (INICIO) ! (LLAMADO) !
              (P VER) ! (LECTURA) ! (ESCRITURA) ! ;
```

```
P(DECL)=(CTE)(L DE CTES) ! (REAL)(4011)(L DE ID) !
        (TEXTO)(4011)(L DE ID) ! (ARREGLO)(4012)(TIPO ARR)(L DE ARREGLOS) !
        (GRAFICA)(4013)(L DE GRAF TAB) ! (TABLA)(4014)(L DE GRAF TAB) !
        (HISTOGRAMA)(4015)(L DE HISTO) ! (ARCHIVO)(L DE AR) !
        (RUTINA)(Z TIPO)(IDENTIF)(ZONA DE PAR FOR)(INICIO) !
        (FORMATO)(L DE FOR);
P(L DE AR)=(IDENTIF)'='(TEXT)(R ARCH) ;
P(R ARCH)=',''(IDENTIF)'='(TEXT)(R ARCH) !  ;
P(L DE CTES)=(DEF DE CTE)(RD1) ;
P(RD1)=','(DEF DE CTE)(RD1) ! ;
P(DEF DE CTE)=(IDENTIF)'='(4010)(DEF);
P(DEF)=(NUM O CONS) ! (TEXT);
P(TIPO ARR)=(REAL)(REN COL) ! (TEXTO) ! (REN COL);
P(REN COL)=(REN) ! (COL) !  ;
P(L DE ARREGLOS)=(DEF ARR)(RD4);
P(RD4)=',' (DEF ARR)(RD4) !  ;
P(DEF ARR)=(L DE ID)'['(L DEF DIM)']';
P(L DE ID)=(IDENTIF)(RD5);
P(RD5)=','(IDENTIF)(RD5) !  ;
P(L DEF DIM)=(LIMITES)(RD55);
P(RD55)=','(LIMITES)(RD55) !  ;
P(LIMITES)=(NUM O CONS) ':' (NUM O CONS);
P(NUM O CONS)=(OP UNARIO)(N O C);
P(N O C)=(NUMERO) ! (IDENTIF) ;
P(L DE GRAF TAB)=(DEF GT)(RD6);
P(RD6)=',' (DEF GT) (RD6) !  ;
P(DEF GT)=(L DE ID)'[' (NUM O CONS) ',' (NUM O CONS) ']';
P(L DE HISTO)=(DEF HIST)(RD8);
P(RD8)=','(DEF HIST)(RD8) !  ;
P(DEF HIST)=(L DE ID) '[' (NUM O CONS) ']';
P(ZONA DE PAR FOR)='('(L DE PAR FOR)')' !  ;
P(L DE PAR FOR)=(ZONA TIPO VAL)(L DE ID)(RD11) ;
P(RD11)=(ZONA TIPO VAL)(L DE ID)(RD11) !  ;
P(ZONA TIPO VAL)=(VAL)(REAL) ! (TIPO) ;
P(TIPO)=(ARREG) ! (REAL) ! (TEXTO) ! (ESTRUCTU) ;
P(ESTRUCTU)=(GRAFICA) ! (TABLA) ! (HISTOGRAMA) ;
P(ARREG)=(ARREGLO)(TIPO ARR) '[' (NO DIM) ']' ;
P(NO DIM)=',' (NO DIM) !  ;
P(Z TIPO)=(REAL) !  ;
P(L DE FOR)=(DEF DE FOR)(RF1);
P(RF1)=',' (DEF DE FOR)(RF1) !  ;
P(DEF DE FOR)=(IDENTIF) '=' (L DE TEX);
P(L DE TEX)=(TEXT)(RLC);
P(RLC)=',' (TEXT)(RLC) !  ;
                  %
                  %************* CONDICIONAL ******************
P(COND)=(P SI)(RC1);
P(RC1)=(SINO)(5046)(P GRAL) !  ;
P(P SI)=(SI)(5033)(EXP)(5034)(ENTONCES)(5044)(P INC);
```

```
                %
                %***********   ITERACIONES ********************
P(ITER)=(REPITE)(CUERPO ITER);
P(CUERPO ITER)=(INICIO) !
                (P INC)(HAS)(TERMINA) !
                (CON)(IDENTIF)':='(EXP)(SUMANDO)(EXP)(HASTA)(EXP)(ESTO)(P INC) !
                (MIENTRAS)(EXP)(ESTO)(P INC) ;
P(HAS)=(HASTA)(EXP);
                %*************      VER      ********************
P(P VER)=(VER)(5043)(NUMERO);
                %**************    LLAMADO     **************
P(LLAMADO)=(IDENTIF)(ZONA PAR);
P(ZONA PAR)=(ZONA DE PAR REALES) !  ;
                %*************    ASIGNACION     ***************
P(ASIG)=(5035)(EST REF) ':=' (5033)(EXP)(5034) ;
P(EST REF)=(IDENTIF)(OPER ESTRUC)(EXP SELEC) ;
P(OPER ESTRUC)= '.' (OP EST) !   ;
P(OP EST)=(TITREN) ! (TITCOL) ! (NOMREN) ! (NOMCOL) !
           (TITULO) ! (MARCOCOL) ! (MARCOREN) ! (CUERPO) ;
              %***********   ENTRADA Y SALIDA  ************
P(LECTURA)=(LEE)(DE)(IDENTIF)(FIN ARCH)(CON)(FORM)(L DE RECEP) ;
P(FIN ARCH)=(FIN) (IDENTIF) !  ;
P(FORM)= (LIBRE) ! (L DE TEX) ! (IDENTIF);
P(L DE RECEP)=(ESTO)(EST REF )(RL3)  !  ;
P(RL3) = ',' (EST REF)(RL3)  !  ;
P(ESCRITURA)=(ESCRIBE)(EN)(IDENTIF)(CON)(FORM)(L DE ELEM A ESC);
P(L DE ELEM A ESC)=(ESTO)(EXP)(RE5) !  ;
P(RE5)= ',' (EXP)(RE5) !  ;
              %********** SELEC    ***********************
P(SELEC)=(LA)(EXP)(DE)(INICIO);
              %****************    EXP     ***************
P(EXP)=(EXP SIMPLE)(R3) ;
P(R3)=(OP DE REL)(5001)(EXP SIMPLE)(5025) !  ;
P(OP DE REL)= '<=' ! '<>' ! '<' ! '=' ! '>=' ! '>' ;
P(EXP SIMPLE)=(OP UNARIO)(5015)(TERMINO)(5026)(R4) ;
P(OP UNARIO)= '+' ! '-' !   ;
P(R4)=(OP SUMA)(5007)(TERMINO)(5025)(R4) !  ;
P(OP SUMA)='+' ! '-' ! (O) ;
P(TERMINO) = (FACTOR)(R5) ;
P(R5)=(OP MULT)(5010)(FACTOR)(5025)(R5) !  ;
P(OP MULT)= '*' ! '/' ! '><' ! (Y) ;
P(FACTOR)=(OPERANDO)(R6);
P(R6)='**' (5014)(OPERANDO)(5025) ! '@' (5019)(5026) ! ''''  (5020)(5026) !   ;
P(OPERANDO)=(5027)(PRIMARIO)(EXP SELEC) ! (NO)(OPERANDO)(5018)(5026) ;
P(PRIMARIO)=(NUMERO)(5028) ! (TEXT)(5029) ! (IDENTIF)(5032)(ACPV) !
            (PI)(5030) ! (E)(5031) ! '(' (EXP) (GRAF O REAL ARR) !
            '!' (EXP) '!' (5021)(5026) ! (ARREGLO EXP) ;
```

```
P(GRAF O REAL ARR)='}' !  '.' (EXP)  ',' (EXP)  ')' ;
P(ACPV)= '.' (OP EST) ! (ZONA DE PAR REALES) !   ;
P(ARREGLO EXP)='[[' (LIMITES EXP)  ':'
                    (LISTA DE EXP REP) ']]' (5037) ;
P(LIMITES EXP)= (5036)(NUM O CONS)(R7) ;
P(R7)= ',' (NUM O CONS) (R7) !   ;
P(LISTA DE EXP REP)=(EXP REP)(R8) ;
P(R8)= ',' (EXP REP) (R8) !   ;
P(EXP REP)= (5038)(NUM O CONS) '(' (LISTA DE EXP REP) ')' (5039) !
            (EXP) (5040) ;
P(EXP SELEC)=(EXP SELEC DE ARR) !   ;
P(EXP SELEC DE ARR)= '[' (5041)(LISTA DE DIM) ']' ;
P(LISTA DE DIM)=(ELEM DE DIM Y PTO) (R12);
P(R12)=',' (ELEM DE DIM Y PTO) (R12) !   ;
P(ELEM DE DIM Y PTO)='.' ! (ELEM DE DIM) !   ;
P(ELEM DE DIM)=(GRUPO DE ELEM) (R13) ;
P(R13)='&' (GRUPO DE ELEM) (R13) !   ;
P(GRUPO DE ELEM) = (EXP)(5034)(INTERVALO);
P(INTERVALO)= '_' (EXP)(5034) !   ;
P(ZONA DE PAR REALES) = '(' (LISTA DE PAR REAL) ')' ;
P(LISTA DE PAR REAL)=(EXP) (R14) ;
P(R14) = ',' (EXP) (R14) !   ;
```

APENDICE B.

Lista de Errores en Compilación.

0. Proposición, declaración u operación que aún no se implanta.

1. Identificador que se está declarando por segunda vez en un mismo bloque.

2. Demasiadas variables en el programa fuente. El arreglo ILI de la Tabla de Símbolos está saturado.

3. Demasiadas variables en el programa fuente. El stack de descriptores (arreglo DES) se desbordó.

4. Texto demasiado largo. Se trunca al máximo tamaño permitido: 60 caracteres.

5. Límite muy grande.

6. Constante no declarada.

7. No es constante real.

8. Declaración de arreglos. Alguna dimensión está mal declarada. El límite inferior es mayor al límite superior.

9. El valor del número o constante debe ser positivo.

10. Aparición de un identificador que no ha sido declarado.

11. ERROR EN EL COMPILADOR.
    No coinciden apuntadores en la Tabla de Símbolos con respecto al arreglo LIG. (VEDESC).

12. ERROR EN EL COMPILADOR.
    Error al tratar de leer un siguiente descriptor en el arreglo DES. (VESIG).

13. Debe haber un identificador de constante.

14. Demasiadas dimensiones en la declaración de un arreglo.
    Máximo = 10.

15. Demasiados parámetros en la declaración de una rutina o función. Máximo = 10.

16. Un arreglo que tenga más de una dimensión no puede declarar se como arreglo renglón.
17. La variable o expresión utilizada como expresión condicional debe ser de tipo real.
18. Expresión demasiado extensa. El stack de operandos se derramó.
19. ERROR DEL COMPILADOR.
El stack de operandos se encuentra vacío.
20. Expresión demasiado extensa. El stack de operadores está lleno.
21. ERROR DEL COMPILADOR.
El stack de operadores está vacío.
22. Se trata de sumar el cuerpo de una gráfica con una estructura de distinto tipo.
23. Combinación inválida de operandos en una operación binaria: una estructura real con una estructura texto.
24. Los operadores unarios (menos unario, negación, inversa, determinante y traspuesta) sólo pueden aplicarse a estructuras de tipo real.
25. Operación binaria inválida. Entre estructuras de tipo texto sólo se puede aplicar la suma (+).
26. Operación unaria inválida. A las estructuras tipo texto no se les puede aplicar operadores unarios.
27. En operaciones binarias sólo pueden participar operandos que sean escalares, arreglos, campos de estructuras con forma de escalar o de arreglo y cuerpos de gráfica.
28. Variable inválida en las expresiones de arreglo explícito. Se permiten escalares, arreglos y campos de estructuras que sean escalares o arreglos.
29. A un operando que no es arreglo, se le trata de aplicar una expresión selectiva de arreglo.
30. Declaración de arreglo. El límite inferior de todas las dimensiones debe ser uno.

31. Del lado izquierdo de una proposición de asignación no puede aparecer una constante.

32. Declaración de arreglo. El tamaño de cada dimensión debe ser al menos de dos.

33. No es permitido que un arreglo explícito aparezca dentro de otro arreglo explícito.

34. Demasiadas expresiones para constituir a un arreglo explícito. Eliminar expresiones o simplificarlas.

35. No se permite combinar expresiones de tipo real con expresiones de tipo texto, dentro de un mismo arreglo explícito.

36. Demasiadas expresiones en la expresión selectiva de arreglo. Eliminar expresiones o simplificarlas.

37. En la expresión selectiva de arreglo aplicada a una variable a la que se le va a asignar una expresión, no se permite el uso del operador punto. Se sustituye por el operador vacío.

38. Se trata de aplicar una expresión selectiva de arreglo a una estructura que no es arreglo.

39. Dentro del arreglo explícito, la repetición de expresiones debe ser positiva.

40. Expresión selectiva de arreglo. Se trata de tener acceso a un elemento negativo o cero.

41. Expresión selectiva de arreglo. Como índices sólo se permiten expresiones compuestas de escalares reales y componentes escalares de arreglos reales.

42. Expresión selectiva de arreglo. No se permiten arreglos de reales, como índices de selección.

43. Es inválido asignar una expresión de tipo real a una variable de tipo texto y viceversa.

44. El tamaño de cada dimensión en las estructuras Tabla y Gráfica debe ser mayor a uno. Se modifica a dos.

45. El número de frecuencias en la declaración de una estructura Histograma debe ser mayor a uno. Se modifica a dos.
46. Se trata de tener acceso a un campo de una variable que no es Tabla, Gráfica, ni Histograma.
47. Se trata de tener acceso a un campo que no existe en ese tipo de estructura.
48. Se trata de asignar una variable de tipo estructura o que es cuerpo de gráfica a otra variable, que es de distinto tipo o viceversa.
49. Etiqueta que fué utilizada, pero que nunca apareció en el bloque. Sólo se pueden hacer transferencias incondicionales estrictamente locales.
50. Error fatal. Demasiadas etiquetas en el programa. Tabla para manejar etiquetas saturadas.
51. Demasiadas etiquetas en el bloque. Se permiten por bloque hasta 10 etiquetas.
52. Demasiadas proposiciones en el bloque. El stack de proposiciones se derramó.
53. Faltan proposiciones de "INICIO" o hay proposiciones mal construidas. Stack de proposiciones vacío.
54. Se trata de dar dos alternativas a una sola condicional: SI <Condición> ENTONCES...SINO...SINO... .
55. Demasiadas proposiciones de "FIN" o falta el "INICIO" para alguna proposición.
56. Final inesperado del programa o bloque. Quedan proposiciones de "INICIO" pendientes.
57. Terminación inesperada del programa fuente. No se encontró la proposición de "FIN" que concluye el programa.
90. No se permite declaración alguna, después de que ya ha aparecido una proposición.
91. Falta la proposición "INICIO", con que debe comenzar el programa.

92. Proposición "INICIO" mal utilizada. Sólo se emplea en el comienzo del programa fuente, de una rutina o de una proposición compuesta.

99. Valor numérico a convertir a entero, resulta demasiado grande. Se toma el máximo valor permitido: + - 32767.

100. Valor numérico real demasiado grande. El máximo exponente permitido para representar un real es 10^{37}.

Los siguientes errores corresponden exactamente al análisis lexicográfico-sintáctico.

101. Caracter inválido en el programa fuente. El conjunto de caracteres permitidos en el código ASCII está dado por el grupo {32,...,126}.

102. Error de sintaxis.

103. Error Fatal. La frase es demasiado grande. El buffer para almacenar al texto limpio, se saturó.

104. Error de sintaxis en un número.

105. Error Fatal. Demasiados identificadores, textos y números en el programa fuente. El arreglo CHARS del diccionario de átomos quedó saturado.

106. Error Fatal. Usar menos identificadores, textos y números en el programa. Al almacenar un número en el arreglo CHARS, éste se derramó.

107. Error Fatal. Demasiados identificadores y números en el programa fuente. El arreglo de apuntadores (VARDIC) del diccionario de átomos se llenó.

108. Error Fatal. Frase sintáctica muy grande. El árbol sintáctico resultante es demasiado grande.

APENDICE C.

Lista de Errores en Ejecución.

Los errores fatales (que cancelan la ejecución del programa obje to) están marcados con un asterisco (*).

0. Característica aún no implantada.

6. AVISO: Se está asignando un arreglo-renglón a un arreglo-columna o viceversa. Se hace la asignación y continua la ejecución del programa.

*7. Se trata de asignar un arreglo a otro que tiene distinto nú mero de dimensiones.

*8. Proposición de asignación. La expresión a asignar y el ope rando a recibirla, son de distinto tipo.

*9. Se trata de asignar a una estructura de tipo escalar, otra estructura que no es escalar.

*10. Es inválido sacar el determinante a un arreglo que no sea una matriz cuadrada de dos dimensiones.

*11. Es inválido sacar la inversa a un arreglo de más de dos dimensiones.

*12. Es inválido sacar la traspuesta a un arreglo de más de dos dimensiones.

*13. Se trata de operar con dos arreglos de distinto número de dimensiones.

*14. Las matrices a multiplicar no son conformables (el número de columnas de la matriz izquierda es distinto al número de renglones de la matriz derecha).

*15. No se permite multiplicar matrices que sean de más de dos dimensiones.

*18. Expresión demasiado grande. Stack de descriptores de operandos saturado.

*19. Error en la biblioteca del programa objeto. Stack de descriptores de operandos vacío.

*20. Es inválido tener acceso a un subarreglo de una estructura

de tipo escalar, en una expresión o en una asignación.

*21. El número de dimensiones en una expresión selectiva de arreglo es distinto al que contiene el operando, al que se va a aplicar la expresión.

22. Los índices a arreglos deben ser escalares reales o componentes escalares de arreglos reales. Se toma como índice al límite superior de la dimensión.

23. Indice a arreglo que es menor a 1. Se toma a 1 como el índice.

24. Indice a arreglo, mayor al límite superior de la dimensión. Se toma dicho límite.

25. Pareja de índices a arreglo, es inválida. El primer elemento debe ser menor al segundo. El segundo elemento se iguala al primero.

26. La concatenación de dos textos excede la capacidad de una variable texto (60 caracteres). Se trunca el texto resultante.

27. La expresión condicional debe ser de tipo real. Se toma como falsa la evaluación de la expresión.

97. Valor númerico que es mayor al máximo entero posible: ± 32767. Se toma al máximo valor posible.

99. División entre cero. Se toma como resultado al valor 9.999999E+30.

100. Potencia A^B con parámetros erroneos.

Si $A < 0$ y $B <> 0$ entonces se hace $(-A)^B$.

Si $A = 0$ y $B <= 0$ entonces se toma como resultado a cero.

APENDICE D

Lista de llamados Semánticos.

En primer lugar se presentan los llamados que se efectúan desde el programa principal del Analizador Semántico.

Llamado	Rutina Ejecutada	Función
4010	DECCTE	Declara una constante real o texto
4011	DECEST	Declara una lista de variables escalares
4012	DECARREGLOS	Declara una lista de arreglos
4013	DECEST(false)	Declara una lista de Gráficas
4014	DECEST(true)	Declara una lista de Tablas
4015	DECHIST	Declara una lista de Histogramas
5033	PROCEXP	Procesa una expresión
5035	ASIGNACION	Procesa una proposición de asignación
5042	DECET	Analiza la aparición de una Etiqueta
5043	BRINCA	Procesa una proposición de transferencia incondicional "GOTO"
5044	CONDICIONAL	Procesa la primera parte de una proposición condicional: SI <Cond> ENTONCES...
5045	SECCONDICIONAL	Se llama al encontrar una proposición de "FIN". Por el momento sólo se procesa cuando corresponde a una condicional o al fin del programa fuente
5046	ALTCONDICIONAL	Procesa la alternativa de una proposición condicional: SINO...
5047	TERCONDICIONAL	Concluye las condicionales sencillas anidadas que queden pendientes al encontrar un fin de proposición ";" .

Los siguientes llamados son hechos dentro de la rutina PROCEXP que procesa expresiones.

Llamado	Rutina Ejecutada	Función
5001 5007 5010 5014 5015 5018 5019 5020 5021	MOPDOR	Estos nueve llamados introducen al stack de operadores un elemento que corresponde a una operación unaria o a una binaria.
5025	OPBINARIA	Toma dos operandos y un operador de los stacks respectivos y procesa una operación binaria
5026	OPUNARIA	Toma un operando y un operador y procesa una operación unaria
5027	VALIDAOPNDO	Valida que una expresión selectiva de arreglo se aplique a un objeto válido
5028 5029 5030 5031 5032	METEOPNDO	Inserta un operando al stack respectivo. El operando puede corresponder a un número, una cadena, a la constante π, a la constante e o a un identificador, respectivamente
5033	IOPNDO: = 0; IOPDOR: = 0;	Este llamado se hace también desde el programa principal de la semántica. En este caso inicializa a los stacks de operandos y de operadores
5034		Finaliza el análisis de una expresión
5036	INIARREXP	Inicia el análisis de un arreglo explícito, procesando las dimensiones que lo componen
5037	ARREXPFIN	Finaliza el análisis de un arreglo explícito
5038	ARREXPNUM	Procesa la repetición de expresiones dentro de un arreglo explícito
5039	ARREXPFNUM	Finaliza el análisis de repetición de expresiones en un arreglo explícito
5040	ARREXPEXP	Analiza una expresión dentro de un arreglo explícito
5041	EXPSELEC	Procesa una expresión selectiva de arreglo

APENDICE D.

A continuación se presentan tres ejemplos de programas que han sido procesados por el compilador estadístico.

A lo largo de los ejemplos se muestran los principales elementos del lenguaje que ya han sido implantados.

Para el tercer ejemplo, se incluye la reseña del análisis semántico (la cual se prende con la instrucción $$D+ y se apaga con $$D-) y también se presenta el programa objeto generado.

EJEMPLO 1

```
%   ESTE PROGRAMA MUESTRA ALGUNAS HERRAMIENTAS DEL LENGUAJE ESTADISTICO
  QUE SE HAN YA IMPLANTADO;

INICIO
 CONSTANTE TAMEDADES=100,
           NUMDATOS=10,
           EDADINICIAL=3;

 ARREGLO EDADES [1:2, 1: TAMEDADES ],  % EDAD CONTIENE LAS EDADES DE UNA
                     POBLACION IMAGINARIA, LAS MUJERES OCUPAN EL RENGLON UNO
                     Y LOS HOMBRES EL DOS;
         SECEDADES [1:2, 1: TAMEDADES ];
 REAL I, J, ED,
      PORTOTAL, PORMUJERES, PORHOMBRES, PORRANGO, EDADRANGO,
      EDADMUJERES, MEDIA, DESVIACION;


 % INICIALIZO EL ARREGLO A CEROS. --NO ES NECESARIO---  ;
 EDADES:=[[2, 100 : 200(0) ]];

 % GENERO Y ASIGNO EDADES, PUES TODAVIA NO SE IMPLANTA LA PROPOSICION
   DE LECTURA ;
 I:=1; J:=1; ED:=EDADINICIAL;
 10 : EDADES[I,J]:=ED;
      SI ED<=20 ENTONCES ED:=-ED*5+100
        SINO ED:=ED-19;
      SI J<NUMDATOS ENTONCES
        J:=J+1
      SINO INICIO
        J:=1; I:=I+1;
      FIN;
      SI I<=2 ENTONCES VER 10;
```

```
% SI EDADES[I,J]=0 ENTONCES CONSIDERO QUE ESE ELEMENTO NO PERTENECE A LA
  POBLACION;

% NO. TOTAL DE HABITANTES;
POBTOTAL:=(EDADES>0)[.,.];

% NO. TOTAL DE MUJERES Y HOMBRES;
POBMUJERES:=(EDADES[1, ]>0)[.];
POBHOMBRES:=(EDADES[2, ]>0)[.];

% PARTE DE LA POBLACION QUE TIENE ENTRE 5 Y 90 A#OS DE EDAD ;
SEGEDADES:=(EDADES>=5) Y (EDADES<=90); % SI SEGEDADES[I,J]=1 ENTONCES
                           EL HABITANTE RESPECTIVO ESTA EN ESE RANGO DE EDAD.
                           SI ES 0, ENTONCES NO ESTA ;

% NO. DE GENTES QUE TIENEN ENTRE 5 Y 90 A#OS;
POBRANGO:=SEGEDADES[.,.];

% SUMA DE LAS EDADES DE LAS PERSONAS DE 5 A 90 A#OS ;
EDADRANGO:=(SEGEDADES*EDADES) [.,.];

% MEDIA Y DESVIACION ESTANDAR DE LA SUBPOBLACION ANTERIOR ;
MEDIA:=EDADRANGO/POBRANGO;
DESVIACION:=(((SEGEDADES*(EDADES-MEDIA))**2)[.,.] / POBRANGO)**0.5;

% SUMA DE LAS EDADES DE TODAS LAS MUJERES ;
EDADMUJERES:=((EDADES[2, ]>0)*EDADES[2, ]) [.];

FIN;
```

EJEMPLO 2

```
INICIO

% LA PRECIPITACION ANUAL ES MEDIDA ANUALMENTE DURANTE 10 AÑOS Y ESTA
 MEDIDA SE EXAMINA PARA VER SI LA CANTIDAD DE PRECIPITACION TIENDE A DE-
 CRECER O A CRECER. LO QUE SE DESEA PROBAR ES LA HIPOTESIS DE INDEPENDENCIA
 ENTRE LA CANTIDAD DE PRECIPITACION Y EL AÑO USANDO LA PRUEBA RHO DE
 SPEARMAN.

     HO : PRECIPITACION Y AÑO SON INDEPENDIENTES.
     HA : EXISTE TENDENCIA A APAREAR VALORES GRANDES DE PRECIPITACION
          CON VALORES GRANDES DE AÑOS O VICEVERSA.

     SEA LA ALFA USADA : ALFA = 0.05 Y LOS CUANTILES PARA N=10 SON
   W025=-0.6364 Y W975=0.6364.


   CONSTANTE MAXDATOS=100,
             ALFA= 0.05,
             W025= -0.6364,
             W975= 0.6364;

   REAL NUMDATOS, VALORRHO, I, J, K;

   ARREGLO PRECIPITACION, RANGOSP, RANGOSA, ANOS [ 1: MAXDATOS ];

   TEXTO RES;

   %  ASIGNA DATOS.--AUN NO HAY PROPOPSICION DE LECTURA--  ;
   I:=1;
   J:=23.56; K:=345;
   NUMDATOS:=10;

   10 :
   PRECIPITACION[I]:=J;
   ANOS[I]:=K;
   I:=I+1;
   J:=J*3.44; K:=K-2;
   SI I<=NUMDATOS ENTONCES VER 10;

   % ASIGNO RANGOS A PRECIPITACION :  ;
   I:=1;
   30 :
   RANGOSP[I]:=(PRECIPITACION[I]>PRECIPITACION[1_NUMDATOS])[.]+
             ((PRECIPITACION[I]=PRECIPITACION[1_NUMDATOS])[.]+1)/2;
   I:=I+1;
   SI I<=NUMDATOS ENTONCES VER 30;
```

```
% ASIGNO RANGOS A AÑOS :   ;
I:=1;
40 :
RANGOSA[I]:=(ANOS[I]>ANOS[1_NUMDATOS])[.]+
            ((ANOS[I]=ANOS[1_NUMDATOS])[.]+1)/2;
I:=I+1;
SI I<=NUMDATOS ENTONCES VER 40;

% CALCULO EL VALOR DE RHO ;
VALORRHO:=1-6*(((RANGOSP[1_NUMDATOS]-RANGOSA[1_NUMDATOS])**2)[.] /
      (NUMDATOS*(NUMDATOS**2-1)));

SI (VALORRHO<W025) O (VALORRHO>W975) ENTONCES
  RES:="SE RECHAZA HO "
SINO
  RES:="SE ACEPTA HO";

FIN;
```

EJEMPLO 3

```
INICIO

%  SE QUIERE AJUSTAR UN MODELO DE LA FORMA
    Y = ALFA + BETA * X + E
  A UN CONJUNTO DE DATOS.

   1.- OBTENER LOS ESTIMADORES DE ALFA Y BETA.
   2.- OBTENER EL COEFICIENTE DE DETERMINACION.
   3.- OBTENER LA TABLA DE ANALISIS DE VARIANZA ( ANDEVA ).
;

  CONSTANTE MAXDATOS=100;

  REAL NUMDATOS, ALFAEST, BETAEST, R2, XMED, YMED,
       SUMACUADREG, SUMACUADRESID, SUMACUADTOT, I, J, K;

  ARREGLO X1, Y1[1 : MAXDATOS];

  TABLA ANDEVA[3,4];

  %  ASIGNA DATOS.--AUN NO HAY PROPOPSICION DE LECTURA--  ;
  I:=1;
  J:=123.45; K:=45.66;
  NUMDATOS:=10;

  $$D+
  10 :
  X1[I]:=J;
  Y1[I]:=K;
  I:=I+1;
  J:=J*(-3.44); K:=K*2.456;
  SI I<=NUMDATOS ENTONCES VER 10;

  % SE CALCULAN LOS ESTIMADORES DE ALFA Y BETA ;

  XMED:=(X1[1_NUMDATOS])[.] / NUMDATOS;
  $$D-
  YMED:=(Y1[1_NUMDATOS])[.] / NUMDATOS;
  BETAEST:=((Y1[1_NUMDATOS]*X1[1_NUMDATOS])[.]-NUMDATOS*XMED*YMED) /
           ((X1[1_NUMDATOS]-XMED)**2)[.];
  ALFAEST:=YMED-BETAEST*XMED;
```

```
% SE CALCULA EL COEFICIENTE DE DETERMINACION ;

 SUMACUADREG:=(((ALFAEST+BETAEST*X1[1_NUMDATOS]) -YMED)**2)[.];
 SUMACUADTOT:=((Y1[1_NUMDATOS]-YMED)**2)[.];
 R2:=SUMACUADREG / SUMACUADTOT ;

 % SE ARMA LA TABLA DE ANDEVA ;
 $$D+
 ANDEVA.TITULO:="TABLA DE ANDEVA";
 ANDEVA.NOMCOL:=[[4 : "GRADOS DE LIBERTAD", "SUMA DE CUADRADOS",
                 "CUADRADOS MEDIOS", "F" ]];
 $$D-
 ANDEVA.TITREN:="FUENTE DE VARIACION";
 ANDEVA.NOMREN:=[[ 3 : "REGRESION", "RESIDUAL", "TOTAL" ]];
 ANDEVA.CUERPO[ ,1]:=[[ 3 : 1, NUMDATOS-2, NUMDATOS-1 ]];
 SUMACUADRESID:=((Y1-(ALFAEST+BETAEST*X1[1_NUMDATOS]))**2)[.];
 ANDEVA.CUERPO[ ,2]:=[[ 3 : SUMACUADREG, SUMACUADRESID, SUMACUADTOT ]];
 ANDEVA.CUERPO[1_2,3]:=[[ 2 : SUMACUADREG, (SUMACUADRESID/(NUMDATOS-2)) ]];
 ANDEVA.CUERPO[1,4]:=(SUMACUADREG*(NUMDATOS-2))/SUMACUADRESID;

FIN; % DEL PROGRAMA ;
```

```
APARICION DE UNA ETIQUETA
ETIQUETA: INDICE 1  NOMBRE 10
INICIO DE ASIGNACION. ANALISIS DEL LADO DERECHO
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :  IOPDOR 0 OPERACION MASUN
VE DESCRIPTOR   : DIRVDT 57 LIGA 12 INFO 56
 DESCRIPTOR DE  57 :  CP 1 CS 4 CT 11 CC 0
METE OPERANDO :  IOPNDO 0 TIPO 2 LOC. 11 NIV 1
OPERACION UNARIA
SACA OPERADOR :  IOPDOR 0 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
ANALISIS DE LA PARTE IZQUIERDA DE LA ASIGNACION
VE DESCRIPTOR   : DIRVDT 59 LIGA 14 INFO 66
 DESCRIPTOR DE  59 :  CP 1 CS 17 CT 13 CC 1
METE OPERANDO :  IOPNDO 1 TIPO 4 LOC. 13 NIV 1
INICIO DE EXP. SECTIVA DE ARREGLO
SACA OPERANDO :  IOPNDO 1 TIPO 4 LOC. 13 NIV 1
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :  IOPDOR 0 OPERACION MASUN
VE DESCRIPTOR   : DIRVDT 56 LIGA 11 INFO 51
 DESCRIPTOR DE  56 :  CP 1 CS 4 CT 10 CC 0
METE OPERANDO :  IOPNDO 1 TIPO 2 LOC. 10 NIV 1
OPERACION UNARIA
SACA OPERADOR :  IOPDOR 0 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
SACA OPERANDO :  IOPNDO 1 TIPO 2 LOC. 10 NIV 1
METE OPERANDO :  IOPNDO 1 TIPO 4 LOC. 13 NIV 1
FIN DE EXP. SELECTIVA DE ARREGLO
3 ELEMENTOS GRABADOS A SELEC
SACA OPERANDO :  IOPNDO 1 TIPO 4 LOC. 13 NIV 1
SACA OPERANDO :  IOPNDO 0 TIPO 2 LOC. 11 NIV 1
FIN DE ASIGNACION
TOPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
INICIO DE ASIGNACION. ANALISIS DEL LADO DERECHO
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :  IOPDOR 0 OPERACION MASUN
VE DESCRIPTOR   : DIRVDT 58 LIGA 13 INFO 61
 DESCRIPTOR DE  58 :  CP 1 CS 4 CT 12 CC 0
METE OPERANDO :  IOPNDO 0 TIPO 2 LOC. 12 NIV 1
OPERACION UNARIA
SACA OPERADOR :  IOPDOR 0 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
ANALISIS DE LA PARTE IZQUIERDA DE LA ASIGNACION
VE DESCRIPTOR   : DIRVDT 60 LIGA 15 INFO 76
 DESCRIPTOR DE  60 :  CP 1 CS 17 CT 115 CC 1
METE OPERANDO :  IOPNDO 1 TIPO 4 LOC. 115 NIV 1
INICIO DE EXP. SECTIVA DE ARREGLO
SACA OPERANDO :  IOPNDO 1 TIPO 4 LOC. 115 NIV 1
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :  IOPDOR 0 OPERACION MASUN
VE DESCRIPTOR   : DIRVDT 56 LIGA 11 INFO 51
 DESCRIPTOR DE  56 :  CP 1 CS 4 CT 10 CC 0
METE OPERANDO :  IOPNDO 1 TIPO 2 LOC. 10 NIV 1
OPERACION UNARIA
SACA OPERADOR :  IOPDOR 0 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
SACA OPERANDO :  IOPNDO 1 TIPO 2 LOC. 10 NIV 1
METE OPERANDO :  IOPNDO 1 TIPO 4 LOC. 115 NIV 1
FIN DE EXP. SELECTIVA DE ARREGLO
3 ELEMENTOS GRABADOS A SELEC
SACA OPERANDO :  IOPNDO 1 TIPO 4 LOC. 115 NIV 1
SACA OPERANDO :  IOPNDO 0 TIPO 2 LOC. 12 NIV 1
FIN DE ASIGNACION
```

```
TOPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
INICIO DE ASIGNACION. ANALISIS DEL LADO DERECHO
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
VE DESCRIPTOR    : DIRVDT 56 LIGA 11 INFO 51
 DESCRIPTOR DE  56 :  CP 1 CS 4 CT 10 CC 0
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. 10 NIV 1
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MASUN
METE OPERADOR :   IOPDOR 0 OPERACION SUMA
METE OPERANDO :   IOPNDO 1 TIPO 1 LOC.  1.00000
OPERACION BINARIA
SACA OPERADOR :   IOPDOR 0 OPERACION SUMA
SACA OPERANDO :   IOPNDO 1 TIPO 1 LOC.  1.00000
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. 10 NIV 1
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. -1 NIV -8420
FIN DE EXPRESION ARITMETICA
ANALISIS DE LA PARTE IZQUIERDA DE LA ASIGNACION
VE DESCRIPTOR    : DIRVDT 56 LIGA 11 INFO 51
 DESCRIPTOR DE  56 :  CP 1 CS 4 CT 10 CC 0
METE OPERANDO :   IOPNDO 1 TIPO 2 LOC. 10 NIV 1
SACA OPERANDO :   IOPNDO 1 TIPO 2 LOC. 10 NIV 1
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. -1 NIV -8420
FIN DE ASIGNACION
TOPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
INICIO DE ASIGNACION. ANALISIS DEL LADO DERECHO
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
VE DESCRIPTOR    : DIRVDT 57 LIGA 12 INFO 56
 DESCRIPTOR DE  57 :  CP 1 CS 4 CT 11 CC 0
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. 11 NIV 1
METE OPERADOR :   IOPDOR 1 OPERACION MULT
METE OPERADOR :   IOPDOR 2 OPERACION MENOSUN
METE OPERANDO :   IOPNDO 1 TIPO 1 LOC.  3.44000
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 2 OPERACION MENOSUN
SACA OPERANDO :   IOPNDO 1 TIPO 1 LOC.  3.44000
METE OPERANDO :   IOPNDO 1 TIPO 2 LOC. -1 NIV -8420
OPERACION BINARIA
SACA OPERADOR :   IOPDOR 1 OPERACION MULT
SACA OPERANDO :   IOPNDO 1 TIPO 2 LOC. -1 NIV -8420
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. 11 NIV 1
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. -1 NIV -8420
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
ANALISIS DE LA PARTE IZQUIERDA DE LA ASIGNACION
VE DESCRIPTOR    : DIRVDT 57 LIGA 12 INFO 56
 DESCRIPTOR DE  57 :  CP 1 CS 4 CT 11 CC 0
METE OPERANDO :   IOPNDO 1 TIPO 2 LOC. 11 NIV 1
SACA OPERANDO :   IOPNDO 1 TIPO 2 LOC. 11 NIV 1
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. -1 NIV -8420
FIN DE ASIGNACION
TOPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
INICIO DE ASIGNACION. ANALISIS DEL LADO DERECHO
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
VE DESCRIPTOR    : DIRVDT 58 LIGA 13 INFO 61
 DESCRIPTOR DE  58 :  CP 1 CS 4 CT 12 CC 0
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. 12 NIV 1
METE OPERADOR :   IOPDOR 1 OPERACION MULT
METE OPERANDO :   IOPNDO 1 TIPO 1 LOC.  2.45600
OPERACION BINARIA
SACA OPERADOR :   IOPDOR 1 OPERACION MULT
SACA OPERANDO :   IOPNDO 1 TIPO 1 LOC.  2.45600
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. 12 NIV 1
```

```
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. -1 NIV -8420
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
ANALISIS DE LA PARTE IZQUIERDA DE LA ASIGNACION
VE DESCRIPTOR    : DIRVDT 58 LIGA 13 INFO 61
 DESCRIPTOR DE  58 :  CP 1 CS 4 CT 12 CC 0
METE OPERANDO :   IOPNDO 1 TIPO 2 LOC. 12 NIV 1
SACA OPERANDO :   IOPNDO 1 TIPO 2 LOC. 12 NIV 1
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. -1 NIV -8420
FIN DE ASIGNACION
TOPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
VE DESCRIPTOR    : DIRVDT 56 LIGA 11 INFO 51
 DESCRIPTOR DE  56 :  CP 1 CS 4 CT 10 CC 0
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. 10 NIV 1
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MASUN
METE OPERADOR :   IOPDOR 0 OPERACION MENOIG
METE OPERADOR :   IOPDOR 1 OPERACION MASUN
VE DESCRIPTOR    : DIRVDT 47 LIGA 2 INFO 6
 DESCRIPTOR DE  47 :  CP 1 CS 4 CT 1 CC 0
METE OPERANDO :   IOPNDO 1 TIPO 2 LOC. 1 NIV 1
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 1 OPERACION MASUN
OPERACION BINARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MENOIG
SACA OPERANDO :   IOPNDO 1 TIPO 2 LOC. 1 NIV 1
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. 10 NIV 1
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. -1 NIV -8420
FIN DE EXPRESION ARITMETICA
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. -1 NIV -8420
CONDICIONAL SENCILLA
INSERCION AL STACK DE PROPOSICIONES: IND 1 PROP. 1
BRINCO
ETIQUETA: INDICE 1  NOMBRE 10
TOPE DEL STACK DE PROPOSICIONES: IND 1 PROP. 1
SALE ELEMENTO DEL STACK DE PROPOSICIONES: IND 0
TOPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
INICIO DE ASIGNACION. ANALISIS DEL LADO DERECHO
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
METE OPERADOR :   IOPDOR 1 OPERACION MASUN
VE DESCRIPTOR    : DIRVDT 59 LIGA 14 INFO 66
 DESCRIPTOR DE  59 :  CP 1 CS 17 CT 13 CC 1
METE OPERANDO :   IOPNDO 0 TIPO 4 LOC. 13 NIV 1
INICIO DE EXP. SECTIVA DE ARREGLO
SACA OPERANDO :   IOPNDO 0 TIPO 4 LOC. 13 NIV 1
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 2 OPERACION MASUN
METE OPERANDO :   IOPNDO 0 TIPO 1 LOC.  1.00000
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 2 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
SACA OPERANDO :   IOPNDO 0 TIPO 1 LOC.  1.00000
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 2 OPERACION MASUN
VE DESCRIPTOR    : DIRVDT 47 LIGA 2 INFO 6
 DESCRIPTOR DE  47 :  CP 1 CS 4 CT 1 CC 0
METE OPERANDO :   IOPNDO 0 TIPO 2 LOC. 1 NIV 1
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 2 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
SACA OPERANDO :   IOPNDO 0 TIPO 2 LOC. 1 NIV 1
METE OPERANDO :   IOPNDO 0 TIPO 4 LOC. -1 NIV -8420
```

```
FIN DE EXP. SELECTIVA DE ARREGLO
3 ELEMENTOS GRABADOS A SELEC
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 1 OPERACION MASUN
INICIO DE EXP. SECTIVA DE ARREGLO
SACA OPERANDO :   IOPNDO 0 TIPO 4 LOC. -1 NIV -8420
METE OPERANDO :   IOPNDO 0 TIPO 4 LOC. -1 NIV -8420
FIN DE EXP. SELECTIVA DE ARREGLO
2 ELEMENTOS GRABADOS A SELEC
METE OPERADOR :   IOPDOR 1 OPERACION DIVISION
VE DESCRIPTOR    : DIRVDT 47 LIGA 2 INFO 6
 DESCRIPTOR DE  47 :  CP 1 CS 4 CT 1 CC 0
METE OPERANDO :   IOPNDO 1 TIPO 2 LOC. 1 NIV 1
OPERACION BINARIA
SACA OPERADOR :   IOPDOR 1 OPERACION DIVISION
SACA OPERANDO :   IOPNDO 1 TIPO 2 LOC. 1 NIV 1
SACA OPERANDO :   IOPNDO 0 TIPO 4 LOC. -1 NIV -8420
METE OPERANDO :   IOPNDO 0 TIPO 4 LOC. -1 NIV -8420
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
ANALISIS DE LA PARTE IZQUIERDA DE LA ASIGNACION
VE DESCRIPTOR    : DIRVDT 51 LIGA 6 INFO 26
 DESCRIPTOR DE  51 :  CP 1 CS 4 CT 5 CC 0
METE OPERANDO :   IOPNDO 1 TIPO 2 LOC. 5 NIV 1
SACA OPERANDO :   IOPNDO 1 TIPO 2 LOC. 5 NIV 1
SACA OPERANDO :   IOPNDO 0 TIPO 4 LOC. -1 NIV -8420
FIN DE ASIGNACION
TOPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
INICIO DE ASIGNACION. ANALISIS DEL LADO DERECHO
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
METE OPERANDO :   IOPNDO 0 TIPO 3 LOC. -1 NIV 1
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 0 OPERACION MASUN
FIN DE EXPRESION ARITMETICA
ANALISIS DE LA PARTE IZQUIERDA DE LA ASIGNACION
VE DESCRIPTOR    : DIRVDT 62 LIGA 16 INFO 86
 DESCRIPTOR DE  62 :  CP 5 CS 25 CT 217 CC 9
VE SIGUIENTE   :  AP. DES 91 CP 2 CS 4 CT 217 CC 0
METE OPERANDO :   IOPNDO 1 TIPO 3 LOC. 217 NIV 1
SACA OPERANDO :   IOPNDO 1 TIPO 3 LOC. 217 NIV 1
SACA OPERANDO :   IOPNDO 0 TIPO 3 LOC. -1 NIV 1
FIN DE ASIGNACION
TOPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
INICIO DE ASIGNACION. ANALISIS DEL LADO DERECHO
INICIO DE EXPRESION ARITMETICA
METE OPERADOR :   IOPDOR 0 OPERACION MASUN
INICIO DE ARREGLO EXPLICITO
METE OPERADOR :   IOPDOR 1 OPERACION MASUN
METE OPERANDO :   IOPNDO 0 TIPO 3 LOC. -1 NIV 1
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 1 OPERACION MASUN
SACA OPERANDO :   IOPNDO 0 TIPO 3 LOC. -1 NIV 1
METE OPERANDO :   IOPNDO 0 TIPO 5 LOC. -1 NIV -8420
METE OPERADOR :   IOPDOR 1 OPERACION MASUN
METE OPERANDO :   IOPNDO 1 TIPO 3 LOC. -1 NIV 1
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 1 OPERACION MASUN
SACA OPERANDO :   IOPNDO 1 TIPO 3 LOC. -1 NIV 1
METE OPERADOR :   IOPDOR 1 OPERACION MASUN
METE OPERANDO :   IOPNDO 1 TIPO 3 LOC. -1 NIV 1
OPERACION UNARIA
SACA OPERADOR :   IOPDOR 1 OPERACION MASUN
SACA OPERANDO :   IOPNDO 1 TIPO 3 LOC. -1 NIV 1
METE OPERADOR :   IOPDOR 1 OPERACION MASUN
```

```
PERACION UNARIA
ACA OPERADOR :   IOPDOR 1 OPERACION MASUN
ACA OPERANDO :   IOPNDO 1 TIPO 3 LOC. -1 NIV 1
IN DE ARREGLO EXPLICITO
 ELEMENTOS GRABADOS A ARREXP
O. DE DIMENSIONES: 1
PERACION UNARIA
ACA OPERADOR :   IOPDOR 0 OPERACION MASUN
IN DE EXPRESION ARITMETICA
NALISIS DE LA PARTE IZQUIERDA DE LA ASIGNACION
E DESCRIPTOR    : DIRVDT 62 LIGA 16 INFO 86
DESCRIPTOR DE  62 :   CP 5 CS 25 CT 217 CC 9
E SIGUIENTE  :   AP. DES 91 CP 2 CS 4 CT 217 CC 0
E SIGUIENTE  :   AP. DES 96 CP 2 CS 4 CT 233 CC 0
E SIGUIENTE  :   AP. DES 101 CP 2 CS 4 CT 249 CC 0
E SIGUIENTE  :   AP. DES 106 CP 1 CS 13 CT 265 CC 0
E SIGUIENTE  :   AP. DES 111 CP 3 CS 4 CT 0 CC 0
E SIGUIENTE  :   AP. DES 116 CP 2 CS 17 CT 280 CC 0
E SIGUIENTE  :   AP. DES 121 CP 2 CS 17 CT 330 CC 0
ETE OPERANDO :   IOPNDO 1 TIPO 5 LOC. 330 NIV 1
ACA OPERANDO :   IOPNDO 1 TIPO 5 LOC. 330 NIV 1
ACA OPERANDO :   IOPNDO 0 TIPO 5 LOC. -1 NIV -8420
IN DE ASIGNACION
OPE DEL STACK DE PROPOSICIONES: IND 0 PROP. 0
INIMA MEMORIA DISPONIBLE : 3993
*** ACABO LA COMPILACION ***  0 ERRORES **
```

CODIGO CORRESPONDIENTE AL PROGRAMA DEL EJEMPLO 3

```
(*$S++*)
(*$G+*)
PROGRAM LENGEST;
USES TRANSCEN, COOGRAL, INICOD;
LABEL 1, 2, 3, 4, 5, 6, 7, 8, 9, 10;
PROCEDURE INI11;
BEGIN
INICIALI;
NIVEL:=1;
DEC1ARR(13,1,100);
DEC1ARR(115,1,100);
DEC1ARR(265,2,3);
DEC2ARR(4);
DEC1ARR(280,1,3);
DEC1ARR(330,1,4);
DEC1ARR(396,1,3);
DEC1ARR(401,1,4);
TOPEMEM:=DISPLAY[1]+407;
END; (* INI11 *)
BEGIN
INI11;
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
SELEC[0]:=0;
MUEVE(10,2,1, 1.00000,1);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
SELEC[0]:=0;
MUEVE(11,2,1, 1.23450E2,1);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
SELEC[0]:=0;
MUEVE(12,2,1, 4.56600E1,1);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
SELEC[0]:=0;
MUEVE(1,2,1, 1.00000E1,1);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
  1 : AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
SELEC[2]:=-(DISPLAY[1]+10+1003);
SELEC[3]:=-1000;
SELEC[1]:=1;
SELEC[0]:=1;
NIVELDOS:=1;
MUEVE(13,4,1,11,2);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
SELEC[2]:=-(DISPLAY[1]+10+1003);
SELEC[3]:=-1000;
SELEC[1]:=1;
SELEC[0]:=1;
NIVELDOS:=1;
MUEVE(115,4,1,12,2);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
NIVELUNO:=1;
OPERBIN(SUMA,10,2, 1.00000,1);
SELEC[0]:=0;
MUEVE(10,2,1,-1,2);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
OPERUNA(MENOSUN, 3.44000,1);
NIVELUNO:=1;
OPERBIN(MULT,11,2,-1,2);
SELEC[0]:=0;
MUEVE(11,2,1,-1,2);
```

```
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
NIVELUNO:=1;
OPERBIN(MULT,12,2, 2.45600,1);
SELEC[0]:=0;
MUEVE(12,2,1,-1,2);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
NIVELDOS:=1;
NIVELUNO:=1;
OPERBIN(MENOIG,10,2,1,2);
     (* TERMINO EL PROCESAMIENTO DE LA EXPRESION ***)
EVALCOND(-1,2);
IF CONDRES THEN BEGIN
GOTO 1;
END; (* SI *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
SELEC[2]:=1;
SELEC[3]:=-(DISPLAY[1]+1+1003);
SELEC[1]:=1;
SELEC[0]:=1;
NIVELUNO:=1;
CONSELARR(13,4);
SELEC[1]:=1;
SELEC[2]:=-1001;
SELEC[0]:=1;
CONSELARR(-1,4);
NIVELDOS:=1;
OPERBIN(DIVISION,-1,4,1,2);
SELEC[0]:=0;
MUEVE(5,2,1,-1,4);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
SELEC[2]:=1;
SELEC[3]:=-(DISPLAY[1]+1+1003);
SELEC[1]:=1;
SELEC[0]:=1;
NIVELUNO:=1;
CONSELARR(115,4);
SELEC[1]:-1;
SELEC[2]:=-1001;
SELEC[0]:=1;
CONSELARR(-1,4);
NIVELDOS:=1;
OPERBIN(DIVISION,-1,4,1,2);
SELEC[0]:=0;
MUEVE(6,2,1,-1,4);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
SELEC[2]:=1;
SELEC[3]:=-(DISPLAY[1]+1+1003);
SELEC[1]:=1;
SELEC[0]:=1;
NIVELUNO:=1;
CONSELARR(115,4);
SELEC[2]:=1;
SELEC[3]:=-(DISPLAY[1]+1+1003);
SELEC[1]:=1;
SELEC[0]:=1;
NIVELUNO:=1;
CONSELARR(13,4);
OPERBIN(MULT,-1,4,-1,4);
SELEC[1]:=1;
SELEC[2]:=-1001;
SELEC[0]:=1;
CONSELARR(-1,4);
```

```
CONSELARR( 1,4);
NIVELDOS:=1;
NIVELUNO:=1;
OPERBIN(MULT,1,2,5,2);
NIVELDOS:=1;
OPERBIN(MULT,-1,2,6,2);
OPERBIN(RESTA,-1,4,-1,2);
SELEC[2]:=1;
SELEC[3]:=-(DISPLAY[1]+1+1003);
SELEC[1]:=1;
SELEC[0]:=1;
NIVELUNO:=1;
CONSELARR(13,4);
NIVELDOS:=1;
OPERBIN(RESTA,-1,4,5,2);
OPERBIN(POTENCIA,-1,4, 2.00000,1);
SELEC[1]:=1;
SELEC[2]:=-1001;
SELEC[0]:=1;
CONSELARR(-1,4);
OPERBIN(DIVISION,-1,4,-1,4);
SELEC[0]:=0;
MUEVE(3,2,1,-1,4);
TOPEMEM:=AUXTOP1;  (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM;  (* INICIA EXPRESION O ASIGNACION *)
NIVELDOS:=1;
NIVELUNO:=1;
OPERBIN(MULT,3,2,5,2);
NIVELUNO:=1;
OPERBIN(RESTA,6,2,-1,2);
SELEC[0]:=0;
MUEVE(2,2,1,-1,2);
TOPEMEM:=AUXTOP1;  (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM;  (* INICIA EXPRESION O ASIGNACION *)
SELEC[2]:=1;
SELEC[3]:=-(DISPLAY[1]+1+1003);
SELEC[1]:=1;
SELEC[0]:=1;
NIVELUNO:=1;
CONSELARR(13,4);
NIVELUNO:=1;
OPERBIN(MULT,3,2,-1,4);
NIVELUNO:=1;
OPERBIN(SUMA,2,2,-1,4);
NIVELDOS:=1;
OPERBIN(RESTA,-1,4,6,2);
OPERBIN(POTENCIA,-1,4, 2.00000,1);
SELEC[1]:=1;
SELEC[2]:=-1001;
SELEC[0]:=1;
CONSELARR(-1,4);
SELEC[0]:=0;
MUEVE(7,2,1,-1,4);
TOPEMEM:=AUXTOP1;  (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM;  (* INICIA EXPRESION O ASIGNACION *)
SELEC[2]:=1;
SELEC[3]:=-(DISPLAY[1]+1+1003);
SELEC[1]:=1;
SELEC[0]:=1;
NIVELUNO:=1;
CONSELARR(115,4);
NIVELDOS:=1;
OPERBIN(RESTA,-1,4,6,2);
OPERBIN(POTENCIA,-1,4, 2.00000,1);
SELEC[1]:=1;
SELEC[2]:=-1001;
SELEC[0]:=1;
```

```
CONSELARR(-1,4);
SELEC[0]:=0;
MUEVE(9,2,1,-1,4);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
NIVELDOS:=1;
NIVELUNO:=1;
OPERBIN(DIVISION,7,2,9,2);
SELEC[0]:=0;
MUEVE(4,2,1,-1,2);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
LEC2CAD(405,15);
SELEC[0]:=0;
MUEVE(217,3,1,-1,3);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
LEC2CAD(422,18);
ARREXP[1]:=-ISTOPNDO;
BXARREXP(5,1,66,4);
LEC2CAD(442,17);
ARREXP[2]:=-ISTOPNDO;
LEC2CAD(461,16);
ARREXP[3]:=-ISTOPNDO;
LEC2CAD(479,1);
ARREXP[4]:=-ISTOPNDO;
CONSARREXP(4);
SELEC[0]:=0;
MUEVE(330,5,1,-1,5);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
LEC2CAD(482,19);
SELEC[0]:=0;
MUEVE(233,3,1,-1,3);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
LEC2CAD(503,9);
ARREXP[1]:=-ISTOPNDO;
BXARREXP(5,1,50,3);
LEC2CAD(514,8);
ARREXP[2]:=-ISTOPNDO;
LEC2CAD(524,5);
ARREXP[3]:=-ISTOPNDO;
CONSARREXP(3);
SELEC[0]:=0;
MUEVE(280,5,1,-1,5);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
AXARREXP( 1.00000,1);
BXARREXP(4,1,5,3);
NIVELUNO:=1;
OPERBIN(RESTA,1,2, 2.00000,1);
ARREXP[2]:=-ISTOPNDO;
NIVELUNO:=1;
OPERBIN(RESTA,1,2, 1.00000,1);
ARREXP[3]:=-ISTOPNDO;
CONSARREXP(3);
SELEC[1]:=1;
SELEC[2]:=-1002;
SELEC[4]:=1;
SELEC[5]:=-1000;
SELEC[3]:=1;
SELEC[0]:=2;
MUEVE(265,4,1,-1,4);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
```

```
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
SELEC[2]:=1;
SELEC[3]:=-(DISPLAY[1]+1+1003);
SELEC[1]:=1;
SELEC[0]:=1;
NIVELUNO:=1;
CONSELARR(13,4);
NIVELUNO:=1;
OPERBIN(MULT,3,2,-1,4);
NIVELUNO:=1;
OPERBIN(SUMA,2,2,-1,4);
NIVELUNO:=1;
OPERBIN(RESTA,115,4,-1,4);
OPERBIN(POTENCIA,-1,4, 2.00000,1);
SELEC[1]:=1;
SELEC[2]:=-1001;
SELEC[0]:=1;
CONSELARR(-1,4);
SELEC[0]:=0;
MUEVE(8,2,1,-1,4);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
CXARREXP(2,7,1,1);
BXARREXP(4,1,5,3);
CXARREXP(2,8,1,2);
CXARREXP(2,9,1,3);
CONSARREXP(3);
SELEC[1]:=1;
SELEC[2]:=-1002;
SELEC[4]:=2;
SELEC[5]:=-1000;
SELEC[3]:=1;
SELEC[0]:=2;
MUEVE(265,4,1,-1,4);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
CXARREXP(2,7,1,1);
BXARREXP(4,1,4,2);
NIVELUNO:=1;
OPERBIN(RESTA,1,2, 2.00000,1);
NIVELUNO:=1;
OPERBIN(DIVISION,8,2,-1,2);
ARREXP[2]:=-ISTOPNDO;
CONSARREXP(2);
SELEC[2]:=1;
SELEC[3]:=2;
SELEC[1]:=1;
SELEC[5]:=3;
SELEC[6]:=-1000;
SELEC[4]:=1;
SELEC[0]:=2;
MUEVE(265,4,1,-1,4);
TOPEMEM:=AUXTOP1; (* FIN DE ASIGNACION *)
AUXTOP1:=TOPEMEM; (* INICIA EXPRESION O ASIGNACION *)
NIVELUNO:=1;
OPERBIN(RESTA,1,2, 2.00000,1);
NIVELUNO:=1;
OPERBIN(MULT,7,2,-1,2);
NIVELDOS:=1;
OPERBIN(DIVISION,-1,2,8,2);
SELEC[2]:=1;
SELEC[3]:=-1000;
SELEC[1]:=1;
SELEC[5]:=4;
SELEC[6]:=-1000;
SELEC[4]:=1;
SELEC[0]:=2;
```

APENDICE F)

MANUAL DE REFERENCIA Y OPERACION

Este apéndice tiene como finalidad mostrar como se encuentran organizados todos los archivos que conforman al compilador en un grupo de discos flexibles y en hacer varias indicaciones de como se deben ejecutar y modificar los programas que lo componen. Un requisito indispensable para utilizar el compilador es conocer el sistema operativo y el editor de APPLE PASCAL (A[2]).

La máquina en que está implantado el compilador es una microcomputadora APPLE-II PLUS, cuyas principales características son que cuenta con 64K en memoria principal y con dos unidades de disco flexible de 140K cada uno. Los discos flexibles utilizados son de 5 y 1/4 pulgadas, de un sólo lado y de densidad sencilla. Además se empleó una impresora ATI-ARGOS.

Los nombres de algunos archivos que contienen programas, la gramática y las palabras clave del lenguaje, llevan una terminación numérica (V.gr.: SEMANTICA8), que sirve para indicar el número de versión de dicho programa o archivo.

Cada vez que se modifique algún archivo de los anteriores se deberá actualizar la terminación numérica. En los archivos que se consideren como versiones definitivas se puede eliminar dicha terminación.

El compilador se encuentra distribuido en cinco discos flexibles. A continuación se muestra la función y el contenido de cada disco.

DISCO 1

Sistema Operativo

Contiene el sistema operativo de PASCAL y deberá permanecer siempre en el drive A, mientras se use el compilador.

Dentro de este disco se encuentra la biblioteca general del sis-

tema (SYSTEM.LIBRARY) que contiene las rutinas empleadas por el compilador estadístico y por los programas objeto generados por el mismo.

Contenido:

SYSTEM.APPLE
SYSTEM.MISCINFO
SYSTEM.EDITOR
SYSTEM.FILER
SYSTEM.PASCAL
SYSTEM.LIBRARY

DISCO 2

Programas de Utilería y Analizador Lexicográfico.

En este disco se encuentran los programas fuentes y objetos de las utilerías: Generador de la Tabla Gramatical (HT7) y Generador del Diccionario de Palabras Clave (HPC2), así como del Reconocedor Léxico-Sintáctico (LEXICO).

Contenido:

HT7.TEXT
HT7.CODE
GTO.TEXT
GTO33.TEXT
GRAMGEN
METASIMB
HPC2.TEXT
HPC2.CODE
PALCL7.TEXT
PCCHAR
PCVDT
RELPC.TEXT
LEXICO.TEXT
LEXICO.CODE

El programa HT7 toma como entrada a GTO.TEXT, el cual es una copia del archivo GTO33.TEXT.

Por cada nueva versión de la gramática se deberá cambiar la terminación numérica de GTO33.TEXT y actualizar a GTO.TEXT. Como salida emite a GRAMGEN, a METASIMB y un listado que puede emitirse a cualquier dispositivo de salida (pantalla, impresora o disco).

NOTA: Si los tres archivos de salida son emitidos al mismo disco, puede ocurrir un error de entrada-salida, por lo que se recomienda mandar el listado a otro disco o a otro dispositivo. El error se debe a que en APPLE PASCAL, al crearse un archivo se aparta el máximo espacio libre posible. Si se llega a tomar todo el espacio libre disponible en el disco, al tratar de crear un segundo archivo en el mismo disco, ocurrirá un error de espacio insufiente.

El programa HPC2 toma como entrada a PALCL7.TEXT y emite a PCCHAR, a PCVDT y a un listado que en este disco está contenido en RELPC.TEXT.

DISCO 3

Analizador Semántico

Contiene tanto al programa principal, como a las rutinas de biblioteca que constituyen al Analizador Semántico.

Además contiene al compilador para facilitar las modificaciones al reconocedor.

Contenido:

SYSTEM.COMPILER	-	Compilador
GLOBAL.TEXT	-	UNIT GLOBAL
INIYFIN.TEXT	-	UNIT INIYFIN
DECLARA.TEX	-	UNIT DECLARA
EXPR.UNO.TEXT	-	Rutinas para procesar
EXPR.DOS.TEXT	-	expresiones.
GTOYCOND.TEXT	-	Rutinas para la proposición condicional y para la transferencia incondicional
SEMANTICA8.TEXT	-	Programa Principal
SEMANTICA8.CODE	-	Reconocedor Semántico

En la sección VI.J se puede ver en forma detallada la estructura del reconocedor.

DISCO 4

Rutinas de Biblioteca para los Programas Objeto.

Contiene todas las rutinas de biblioteca (Run-time Support) que utilizan los programas objeto generados por el compilador estadístico.

Contenido:

SYSTEM.COMPILER	-	Compilador
INICIALI.TEXT	-	UNIT INICOD
UCODUNO.TEXT	-	Estos tres archivos contienen
UCODDOS.TEXT	-	a UNIT CODGRAL
UCODTRES.TEXT	-	
UCODUNO.TEXT	-	Código objeto de UNIT CODGRAL.

Una explicación detallada de la estructura del Run-time Support se encuentra en la sección VI.K.

NOTA: Cada vez que se modifique alguna rutina de biblioteca se deberá actualizar la Biblioteca del Sistema Operativo (SYSTEM.LIBRARY, del disco 1).

DISCO 5

Disco de Trabajo

Contiene los objetos de los dos reconocedores, así como los archivos emitidos por los programas de utilería y que emplean aquellos y también contiene al compilador.

La razón de este disco es poder utilizar el compilador y obtener programas objeto que se puedan ejecutar.

Contenido:

LEXICO.CODE
PCCHAR
PCVDT
GRAMGEN

METASIMB
SEMANTICA8.CODE
SYSTEM.COMPILER

Cada vez que se obtengan nuevas versiones de los archivos contenidos en este disco, se deberá actualizar al mismo.

Operación

Para poder emplear el compilador, primero se debe crear el programa fuente, lo cual se hace por medio del editor de PASCAL (A[2]).

A continuación se corre el programa LEXICO y si no hay errores, se ejecuta al SEMANTICA8. En caso contrario, se modifica el programa fuente y se vuelve a correr LEXICO.

El programa LEXICO emite dos archivos: SEMCHARS y SEMVDT al drive A, que son tomados por SEMANTICA8 y por el programa objeto (al ejecutarse).

Al correr el programa SEMANTICA8 se genera el programa seudo-objeto, que puede dirigirse a cualquier disco y al que el usuario puede darle cualquier nombre.

Si el Reconocedor Semántico finaliza con errores, entonces se deberá corregir el programa fuente y volver a repetir este procedimiento, desde la ejecución de LEXICO.

Si no hay errores, entonces se compila al programa seudo-objeto como si fuera cualquier programa normal en PASCAL.

Si ocurre algún error en la compilación de PASCAL, éste puede ocurrir por falta de espacio o por haber excedido la capacidad del compilador.

Para el primer caso se pueden eliminar archivos superfluos (ajenos al compilador estadístico) y para el segundo se puede modificar el programa seudo-objeto, tratando de dividirlo en rutinas.

Habiendo obtenido finálmente el programa objeto, éste ya podrá ejecutarse, teniendo cuidado de que si emplea cadenas de caracteres, el archivo SEMCHARS respectivo deberá estar presente en el drive A, al momento de su ejecución.

BIBLIOGRAFIA.

A1. APPLE PASCAL Language Reference Manual; APPLE COMPUTER, INC.

A2. APPLE PASCAL Operating System Reference Manual; APPLE COMPUTER, INC.

A3. Aho, Alfred V. y Ullman, Jeffrey D.; Principles of Compiler Design; Addison Wesley; 1979.

B1. Bornal, Richard; Understanding and Writing Compilers; Macmillan Press LTD; 1982.

F1. Forsythe, George E.; Malcom, Michael A. y Moler, Cleve. B; Computer Methods for Mathematical Computations; Prentice-Hall; 1977.

G1. García García, Javier; Diseño de un Lenguaje Orientado a Resolver Problemas Estadísticos; Tesis de Licenciatura de Actuario, UNAM; 1983.

G2. Gries, David; Compiler Construction for Digital Computers; John Wiley & Sons; 1971.

G3. Grogono, Peter; Programming in PASCAL; Addison-Wesley; 1980.

H1. Hopcroft, John E. y Ullman, Jeffrey D.; Introduction to Automata Theory, Languages and Computation; Addison-Wesley; 1979.

H2. Harrison, Michael A.; Introduction to Formal Language Theory; Addison-Wesley; 1978.

H3. Hellerman, Herbert y Smith, Ira A., APL/360, Programming and Applications; McGraw-Hill; 1976.

J1. Jensen, Kathleen y Wirth, Niklaus; PASCAL: User Manual and Report; Springer-Verlag; 1978.

L1. Legarreta Garciadiego, Luis; Compiladores; Fundación Arturo Rosemblueth; México, D.F.; 1983.

P1. Pratt, Terrence W.; Programming Languages: Design and Implementation; Prentice-Hall, Inc.; 1975.

W1. Waite, William M. y Goos, Gerhard; Compiler Construction; Springer-Verlag; 1984.