UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

DESARROLLO DE UN LENGUAJE ORIENTADO A RESOLVER PROBLEMAS ESTADISTICOS (TERCERA FASE)

T E S I S

QUE PARA OBTENER EL TITULO DE:

A C T U A R I O

P R E S E N T A :

MARTIN ROMERO MARTINEZ

México, D. F. 1988

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INDICE

OBJETIVO

En la aplicación de la Estadística es frecuente el manejo de datos en cantidades tales que el uso de una computadora, surge como una necesidad natural para estudiantes e investigadores .

Así, para satisfacer esta necesidad existen dos opciones: hacer un programa o usar un paquete de cómputo, cada una con ventajas y desventajas.

La facilidad en el uso de un paquete es una ventaja de éstos: por otro lado, son programas limitados a resolver problemas específicos, de manera que con la diversidad e innovación en las técnicas estadísticas es imposible la existencia de un paquete para cada técnica.

Otro inconveniente para usar un paquete es la dificultad o imposibilidad para ligar su entrada o salida con otro programa, obligando al usuario a repetir la captura de datos o bien a capturar los resultados de un programa o paquete. Por último, si se encuentra el paquete adecuado, es posible que no esté disponible para nuestro equipo de cómputo.

La desventajas mencionadas en el párrafo anterior son comunes para estudiantes e investigadores, pero algo que afecta específicamente a los estudiantes es lo siguiente: el uso de paquetes obliga al estudiante a realizar una serie de pasos, la mayoría de las veces desligados de la Estadística y del problema a resolver, llegando a ocasionar que el alumno se olvide de comprender y dominar las técnicas estadísticas (objetivo importante de los cursos); ademas crean en el alumno una visión limitada de las aplicaciones de la Estadística.

Dentro de la Estadística es común operar con matrices, característica que no poseen la mayoría de los lenguajes de propósito general, orillando al alumno o al investigador a dedicar parte de su tiempo en programar o modificar rutinas que operen con matrices.

También expresar los algoritmos estadísticos con una notación diferente a la utilizada en Estadística dificulta su implantación.

Como se ve, el papel del alumno puede variar desde un analista-programador hasta alguien que sigue intrucciones, en los dos casos el uso de una computadora no contribuye al aprendizaje de la Estadística.

Por las razones anteriores, se hizo necesaria la creación de un lenguaje que facilite la operación de arreglos y tenga una notación semejante a la utilizada en Estadística.

El diseño del lenguaje fue realizado por miembros del grupo de Computación del Departamento de Matemáticas con la colaboración de los profesores del Laboratorio de Estadística, ambos grupos de trabajo pertenecientes a la Facultad de Ciencias.

El inicio del desarrollo del compilador, además de la recopilación y filosofía del lenguaje, fueron los temas tratados en una primera fase. La construcción de: el analizador lexicográfico y sintáctico, la tabla de símbolos y los programas de apoyo, fueron aspectos del compilador implantados por el trabajo mencionado en una minicomputadora NOVA 3/12 del Laboratorio de Estadística, usando como lenguajes de programación a FORTRAN y BASIC. La descripción de esta fase puede ser leída en [G1].

La continuación del desarrollo del compilador constituyó el trabajo realizado en una segunda fase.Este trabajo que comprendió la conversión a PASCAL del compilador en una microcomputadora FRANKLIN ACE e implantación de elementos importantes del lenguaje como son expresiones y las proposiciones condicional, de asignacion y de transferencia incondicional. El desarrollo de la segunda fase se encuentra en [E1].

El objetivo de esta tesis consiste en implantar el resto del lenguaje : las estructuras de control restantes, las proposiciones de entrada salida, rutinas y funciones. De esta manera, el compilador sera un producto que puedan usar estudiantes e investigadores de Estadística.

Durante el inicio de este trabajo se continuó el desarrollo del compilador en las microcomputadoras FRANKLIN ACE, pero se presentaba comúnmente el problema de la falta de memoria. Este hecho motivó el traslado del compilador a las microcomputadoras PC. La descripción de el traslado junto con las ventajas y desventajas que acarreó, son el contenido de las últimas dos secciones del primer capítulo.

Buscando implantar el mayor número de elementos del lenguaje, en algunos momentos se tomaron restricciones que no impediran el uso corriente del compilador. Cuando esto suceda se esbozara una solución.

El resultado de los trabajos mencionados es la implantación de un lenguaje, entre cuyas características podemos mencionar:

a.- Tiene proposiciones de control de flujo y una estructura de bloques, análogas a las de PASCAL y ALGOL.

b.- Permite operar de una manera sencilla y clara con arreglos. Por ejemplo :sumar, multiplicar dividir, es posible obtener el determinante e inversa de matrices cuadradas y multiplicar matrices conformes. Además, soporta la obtención de subarreglos y la suma de los elementos de una dimensión, procesos muy usados en Estadística.

c.- Sus tipos de datos básicos son : real, texto(hasta 60 caracteres) y un tipo especial llamado cuerpo gráfica, que es una estructura de datos donde se representa la imágen de una gráfica.

d.-Las proposiciones de entrada-salida son similares a las del lenguaje FORTRAN, pues permiten el manejo de formatos.Adicionalmente se cuenta con rutinas especiales para la impresión de Gráficas, Tablas e Histogramas.

e.- Los tipos de datos compuestos son: Histograma, Gráfica y Tabla, estructuras de datos comúnmente usadas en Estadística, ademas de arreglos de reales o textos.

f.- El compilador del lenguaje y el código generado son programas escritos en PASCAL, garantizando una alta transportabilidad. Porque el código generado será un programa en PASCAL, el programa que se ha llamado compilador es en sentido estricto un traductor, pues no genera código de maquina.

g.-Las palabras reservadas del lenguaje son palabras del español.

En el primer capitulo de esta tesis se resumirá el trabajo realizado, las caracteristicas y estructuras del compilador, para luego continuar con su desarrollo en capitulos posteriores.

El COMPILADOR

1.1 INTRODUCCION

El propósito de este capitulo es presentar aspectos relevantes del compilador y del lenguaje que permitan la comprensión del presente trabajo.

Como se dijo, este trabajo es la continuacion de dos tesis, la primera donde se definió el lenguaje e inició el desarrollo del compilador y una segunda, la cual continuó la implantación del lenguaje estadistico.

Por razones obvias, la presentación del lenguaje y el compilador no puede ser completa. La versión del compilador que se presentará corresponde a la desarrollada en [E1], la cual difiere del diseño original en algunos aspectos, la enumeración de estos y las razones por las cuales se modificó el diseño de García [G1] pueden encontrarse en [E1].

Por lo demás, el presente trabajo continuará el desarrollo de Esquivel [E1] y seguirá los lineamientos dados por García [G1].

La tarea del programa que se ha llamado compilador, consiste en recibir un programa fuente escrito con el lenguaje estadistico y entregar un programa equivalente en PASCAL.

El compilador está dividido en dos partes, la primera lee el programa fuente y efectúa los análisis lexicográfico y sintáctico y una segunda, encargada de realizar el análisis semántico y la generación del código.

Para efectuar el análisis sintáctico, es decir, verificar que las proposiciones del programa fuente pertenezcan al lenguaje, fue necesario representar la gramática en una tabla y esta tarea es realizada por un programa de apoyo.

En seguida, se presentan los programas de apoyo, para después presentar y describir las dos partes que constituyen al compilador.

1.2 PROGRAMAS DE APOYO

La sintaxis del lenguaje está definida por una gramática libre de contexto y el objetivo del primer programa de apoyo (HT7) es obtener una representación manejable de ésta.

Un segundo programa (HPC2) crea el diccionario de palabras claves, que durante el análisis lexicográfico formará parte del diccionario de átomos.

A continuación se detalla la operación de los programas de apoyo.

Generador de la tabla gramatical (HT7)

Lee del archivo GTO.EST una gramática escrita en BNF y genera los archivos : GRAMGEN, METASIMB y LISTADO.

El archivo GRAMGEN contiene la representación de la gramática y las convenciones para escribirla, se encuentran en el apéndice C.

Una descripción detallada de la estructura y uso de la tabla gramatical puede leerse en [L1] ó [G1].

El archivo METASIMB almacena los nombres de los símbolos no terminales de la gramática. Se usa al escribir el contenido de un árbol sintáctico, pues se escribe con el nombre de cada nodo.

La gramática leida, su representación y el diccionario de símbolos no terminales son el contenido del archivo LISTADO.

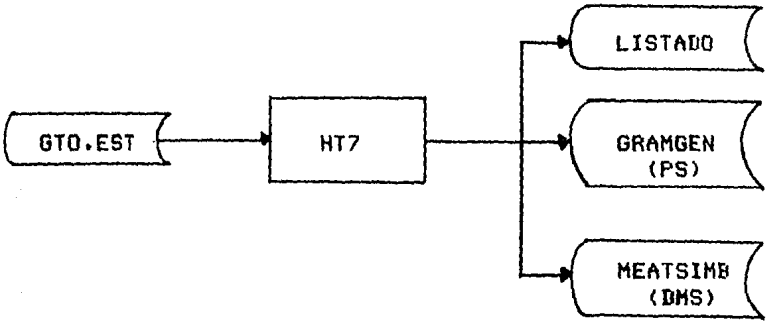


Diagrama 1.1

En la representación gráfica de un archivo, el nombre en paréntesis, indicará el nombre de la estructura de datos en memoria, donde vive (si vive) durante la ejecución del programa.

Generador del diccionario de palabras clave (HPC2)

Lee del archivo PALCL8.EST las palabras claves del lenguaje, separadas por blancos y caracteres fin de línea, crea el diccionario de palabras clave, para luego almacenarlo en los archivos PCVDT y PCCHAR (Ver diagrama 1.2).

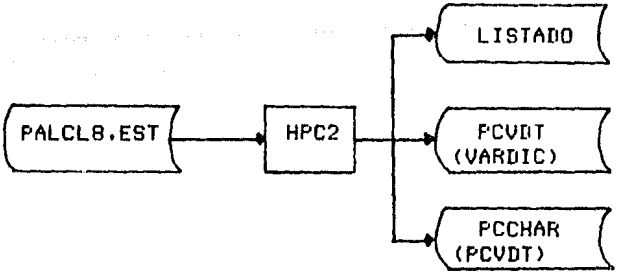


Diagrama 1.2

El diccionario de palabras clave consta de un arreglo de caracteres (CHARS) y otro de enteros (VARDIC). El arreglo CHARS es grabado en PCCHAR y el arreglo VARDIC en PCVDT.

El arreglo CHARS contiene una representación de las palabras clave, formada por su longuitud y los caracteres que la constituyen. Los elementos de VARDIC son apuntadores a CHARS, indicando donde inicia la representación de cada palabra clave.

Los archivos LISTADO podrán dirigirse a disco, impresora o monitor.

PALCL8.EST y GTO.EST son archivos tipo texto de PASCAL, creados con el editor de Turbo Pascal.

1.3 ANALISIS LEXICOGRAFICO Y SINTACTICO

En esta primera parte del compilador, se lee el programa fuente y se obtiene un diccionario, que contiene a los identificadores, textos y números encontrados en el programa leido y un árbol de derivación para cada frase del programa fuente que pudo ser reconocida por el analizador sintáctico.

Los árboles sintácticos construídos son grabados en el archivo ANREC y el diccionario de los átomos es almacenado en los archivos: SEMCHARS y SEMVDT.

Los tres archivos mencionados constituyen la representación del programa fuente que es procesada por la segunda parte del compilador.

El diccionario de palabras claves es usado para reconocer a éstas en el programa fuente, así como la tabla gramatical sirve al analizador sintáctico para identificar las frases del programa fuente pertenecientes a la gramática del lenguaje.

Cuando se imprime un árbol de derivación, los nombres de los nodos son leidos de la tabla de nombres de los símbolos no terminales.

Las tres estructuras mencionadas vivirán en memoria pricipal y son leídas al iniciar el análisis lexicográfico-sintáctico.

Las frases del lenguaje para el analizador sintáctico abarcan hasta un símbolo de punto y coma ';' o la palabra clave 'INICIO'.

El análisis es manejado desde el analizador lexicográfico, el cual lee el programa fuente y lo representa con una notación denominada 'texto limpio' hasta encontrar el fin de una frase; en ese momento se invoca al analizar sintáctico; cuando éste termina su función, regesa el control al analizador lexicográfico para procesar la siguiente frase (Ver figura 1.3).

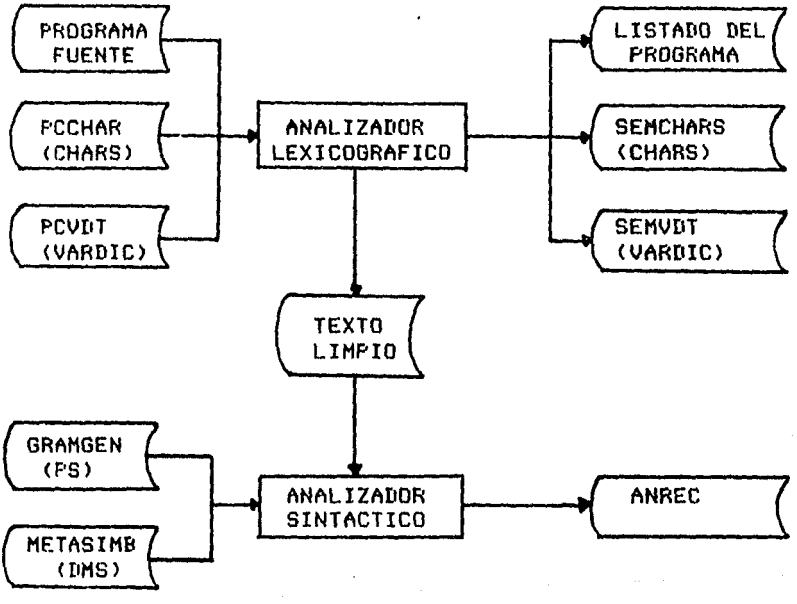


Diagrama 1.3

El listado del programa fuente se numera por frase sintáctica como puede verse en los ejemplos del apendice E.

1.3.1 ANALISIS LEXICOGRAFICO.

El analizador lexicográfico se encarga de reconocer los átomos (tokens) que forman el programa fuente y de manejar las opciones de compilación; reconocido el átomo, se representa por un número o se almacena en el diccionario de átomos y se representa por dos números, la sucesión de números generados y el diccionario de átomos contituyen el texto limpio.

El texto limpio es una representación compacta del programa fuente porque no tiene comentarios o blancos, y es manejable porque los átomos son manejados con su tipo.

El diccionario de átomos está formado por dos arreglos: CHARS, arreglo de caracteres donde vive el átomo y su longitud, y VARDIC arreglo de enteros, cuyos elementos apuntan a CHARS señalando dónde inician las representaciones de los átomos.

Los tipos de átomos reconocidos por el analizador son: identificadores, números, palabras claves, textos(cadenas de caracteres) y caracteres especiales, a continuación se describe el tratamiento dado por el analizador a los diferentes tipos de átomos.

Se identifica un átomo tipo palabra clave o identificador, porque el primer caracter es una letra; se busca el átomo en el diccionario; si no esta presente es almacenado, para luego representarlo en el texto limpio por su tipo e índice del elemento de VARDIC donde vive su apuntador a CHARS.

El diccionario de átomos es inicializado con las palabras clave, leyéndose éstas y su longuitud de los archivos PCVDT y PCCHAR, la primera localidad libre de VARDIC es guardada en IVDTKW. Comparando el lugar ocupado dentro del diccionario con IVDTKW se determina si un átomo es una palabra reservada.

Cuando el primer caracter es comillas ('), se trata de un átomo tipo texto; se almacena comenzando con comillas (') en el diccionario y se representa en el texto limpio por su tipo y un apuntador a CHARS. Cuando dentro de un texto se quiera escribir comillas ('), se deberan escribir dos comillas contiguas.

Si se detecta una opción de compilación para modificar el funcionamiento del analizador lexicografico, ésta se activa o se desactiva; en cambio, si afecta al analizador semántico o al programa generado, se construye y almacena un árbol indicando la opción y sus parámetros. Las opciones de compilación y su significado, pueden consultarse en el apendice A.

Los caracteres especiales son representados por su código ASCII. Cuando el caracter es ';' o se encuentra un 'INICIO' (que no sea el primer átomo) se inserta un punto y coma ';' y es el momenmo de llamar al analizador sintáctico, pues se ha terminado una frase.

Si el átomo encontrado es un número, se guarda y revisa que sea sintácticamente correcto, también, es representado por su tipo y su apuntador a CHARS.

Los tipos de átomos junto con su clave son:

TIPO DE ATOMO	CLAVE
Palabra clave	2000
Identificador	1020
Numero	1010
Texto	1040

1.3.2 ANALISIS SINTACTICO

Durante el análisis sintáctico se revisa que las frases del programa fuente encontradas en el texto limpio pertenezcan al lenguaje. Además se graban en el archivo ANREC los árboles de derivación de las frases sintácticamente correctas.

El analizador sintáctico usa la tabla gramatical (arreglo PS) para reconocer el texto limpio. El representar la gramática del lenguaje en una tabla y usar ésta en el reconocimiento sintáctico, permite modificar a la gramatica (con ciertas restricciones) sin modificar el analizador sintáctico, facilitando la implantación del compilador.

Por la importancia de la gramática y de el analizador sintáctico se presentan a continuación algunas caracteristicas de éstos.

Gramática.

Para facilitar el reconocimiento sintáctico, el analisis se hace frase por frase, de esta menera, a partir del simbolo distinguido se puede derivar una declaración o una proposición simple o parte de una proposicion compuesta. Cuando se tienen proposiciones anidadas o una proposicion compuesta, el análisis se efectuará con uno o varios llamados al reconocedor sintáctico, generando también, uno o varios árboles cuya raiz es el simbolo distingido.

La gramática posee dos tipos de producciones : las tipo B y las tipo P, las producciones tipo P definen un simbolo no terminal en una o varias alternativas, cada una de estas formadas por símbolos terminales y no terminales.

Las producciones tipo B definen un simbolo no terminal con un número entero predefinido (N), algunos de estos números estan relacionados con la representacion de los átomos en el texto limpio, permitiendo reconocer el programa fuente a través del texto limpio.

La interpretación de N es la siguiente:

-Si 2000 < N < 4000 , el átomo esperado es la palabra clave que vive en CHARS a partir de la localidad VARDIC[N-2000].

-Si N es 1020, 1010 o 1040, como se ha dicho representan respectivamente la clave del tipo de los átomos identificador, número y texto.

-Si 4000 <= N <=10000, N representa un llamado semántico, que es un pseudo-apuntador a una o varias rutinas del analizador semántico. El analizador sintáctico crea un nodo con alternativa N.

El reconocedor sintáctico impone las siguientes restricciones a la gramática :

-No debe de tener simbolos muertos (i.e. todos los simbolos deben generar al menos una cadena de simbolos terminales).

-Debe carecer de simbolos inalcanzables (todo simbolo podrá ser derivado desde el simbolo distinguido).

-Ningún simbolo debe derivar en uno o mas pasos, únicamente a él mismo.

-La gramática debe ser recursiva derecha.

En cuanto a las presentación de la gramática deben observarse las siguientes reglas:

-Todas las producciones con igual antecedente deben presentarse juntas.

-Cuando dos producciones tienen antecedentes iguales y el consecuente de una es prefijo de la otra, producción con el consecuente más grande deberá aparecer primero.

La representación de la gramática vivirá en el arreglo PS y es leída al iniciar el análisis lexicográfico- sintáctico.

Analizador Sintáctico.

El reconociemiento sintáctico se hace de arriba a abajo y de izquierda a derecha, aplicando el método de descenso recursivo y tratando de reconocer el texto limpio con la técnica de prueba y error . El análisis es guiado por la tabla gramatical, pues para sustituir un no-terminal, se prueban uno a uno los consecuentes de las producciones con antecedente igual al símbolo

no-terminal que desea reconocer, y precisamente de la tabla gramatical se 'leerán' las producciones. Una presentación detallada del analizador sintáctico puede encontrarse en [L1].

Si el análisis sintáctico es correcto se graba al archivo ANREC una representación lineal del arbol de derivación. Esta representación consta de los arreglos A y NP . El arreglo A, llamado arreglo de alternativas, donde cada elemento contiene el número de alternativa que fue usada en la sustitución o llamado semántico o por un apuntador al diccionario de átomos ; el arreglo NP contiene para cada nodo un apuntador al primer nodo hermano a la derecha o en caso contrario, al nodo ascendente más próximo.

El arreglo auxiliar PN contiene para cada nodo un apuntador a la tabla de simbolos no terminales de la gramática, para imprimir, si se requiere, cada nodo con su nombre. El arreglo PN, no forma parte de la representación del árbol que se pasa al analizador semántico.

1.4 ANALISIS SEMANTICO Y GENERACION DE CODIGO

El analizador semántico se encarga de validar el significado de las proposiciones que forman parte del programa fuente. No se realiza una validación semántica total porque algunas caracteristicas de los operandos temporales son conocidas hasta la ejecución.

Los aspectos revisados por el analizador son: la correspondencia entre operadores y operandos, la declaración de los objetos del lenguaje, determinación del alcance de las delaraciones, y el uso correcto del las estructuras de control.

Al mismo tiempo que se revisan semánticamente las proposiciones del programa fuente, se genera el código necesario. La definición y explicación de las reglas semánticas del lenguaje pueden consultarse en [G1], salvo pequeñas modificaciones, hechas y señaladas en [E1].

En la generación de código se manejan aspectos cómo la administración de memoria en ejecución, la representación de los distintos objetos declarables en el lenguaje y la generación de codigo capaz de realizar las acciones expresadas por el usario a través del programa fuente.

La segunda parte del compilador toma como entrada el diccionario de átomos (archivos SEMCHARS y SEMVDT) y los árboles de derivación (contenidos en el archivo ANREC), entregando como salida, si el análisis semántico es correcto, un archivo que contenga el código generado.Si el análisis semántico es incorrecto se entrega una lista de errores encontrados. Cada aviso de error consta de dos números el número de frase y la clave del error detectado. La lista de errores junto con su clave pueden ser leídas en el apéndice B.

Opcionalmente se muestra un monitoreo del análisis semántico. El monitoreo y la lista de errores son el contenido del archivo LISTADO .

La mayor parte del código generado son llamados a rutinas escritas en PASCAL, estas rutinas viven en los archivos UCODi.PAS con i=0, 1, 2, 3, 5, 6, 7, 8.

Para obtener código ejecutable, el código generado debe ser compilado por Turbo Pascal v3.01A

Es conveniente mencionar que para ejecutar el programa compilado es necesaria la existencia del archivo SEMARCHARS, pues éste, contiene las literales y constantes tipo texto del programa fuente. De no existir SEMCHARS, las literales y constantes tipo texto se consideran nulas.

Para cada uno de los árboles de derivación, representados por los arreglos A y NP, se recorre el árbol hasta encontrar un llamado semántico (un elemento de A mayor a 4000). Dependiendo de la alternativa tomada en el nodo, se llama a la rutina semántica adecuada. La rutina invocada revisa los nodos cercanos al nodo que causó su activación, y en función de las alternativas usadas en los nodos revisados, efectuará las acciones semánticas pertinentes.

En seguida, se describe : La tabla de símbolos, el manejo de memoria y los elementos del lenguaje implantados hasta el momento de iniciar el presente trabajo.

1.4.1 TABLA DE SIMBOLOS.

Con el fin de realizar la validación semántica y la generación de código, es necesario conocer en compilación la información semántica y la ubicación en memoria, si la tienen, de los objetos declarados en el programa fuente.

Las características semánticas mínimas de todo objeto son : tipo, forma y nivel lexicográfico. Para algunos objetos como los arreglos es necesario almacenar más información, como el número de dimensiones y los límites de éstas. Las estructuras de datos donde se guardan las características semánticas y ubicación en memoria de los objetos, se llaman descriptores.

La tabla de símbolos está organizada de manera que facilita el decidir a qué objeto hace referencia un identificador. Esto es importante, porque al tener el lenguaje una estructura de bloques, es posible la existencia de dos objetos distintos asociados a un identificador común.

Los elementos que forman la tabla de símbolos son:

CHARS.-Arreglo de caracteres, donde se almacenan, como se explicó en 1.3, los identificadores asociados a los objetos, las literales numéricas y alfanuméricas.

VARDIC.-Arreglo cuyos elementos son apuntadores a CHARS, indicando los elementos de CHARS, donde inician las representaciones de los átomos.

CHARS y VARDIC son estructuras estáticas durante la segunda fase de compilación.

Otras estructuras son:

DES.-Arreglo donde se guardan : localidad,forma,tipo y otras características de los objetos declarados, el número de celdas que ocupará un objeto en DES dependera de su tipo.Además contiene apuntadores a VARDIC, de manera que es posible conocer el identificador asociado al objeto.

INFO.-Arreglo cuyos elemetos son apuntadorse a DES, indicando donde inician las descripciones de los objetos.

LI.-Arreglo paralelo a INFO donde se almacenan listas. Estas se forman por apuntadores a los elementos de DES, donde viven los descriptores de los objetos asociados a un identificador común.

Asi, INFO[j] apunta al descriptor del objeto (que vive en DES) y LI[j] apunta al siguiente elemento de la lista.

LIG.-Arreglo con el mismo número de elementos de VARDIC, apuntando éstos a las cabezas de las listas que viven en LI

NIVEL.- Variable que contiene el número de nivel lexicográfico que se está procesando.

NIVELS.-Arreglo donde se guardan apuntadores a DES, los cuales sirven para marcar los descriptores que pertenecen a los diferentes niveles lexicográficos.

Por ejemplo, entre NIVELS[1] y NIVELS[2] viven todos los descriptores de los objetos declarados en el primer nivel lexicográfico (programa principal).

Los elementos de NIVELS que no se usan, apuntan hacia afuera de DES.

Los arreglos LI, LIG, INFO, DES son estructuras dinámicas durante la segunda fase de compilación. Con este diseño la búsqueda y declaración de las características semánticas de un objeto es sencilla. Si el identificador del objeto (id) vive en CHARS a partir del elemento VARDIC[J], entonces:

.-Para saber si existe algún objeto desde el lugar actual de compilación asociado a id, basta con ver si LIG[J] es no cero. Entendiendo que, si existe, fue declarado en el bloque actual o en un bloque que contiene al bloque actual.Si LIG[J] es no cero, entonces, INFO[LIG[J]] apunta al lugar de DES a partir de donde vive el descriptor del último objeto declarado cuyo identificador es id.

.-Para declarar un objeto, si no existe objeto asociado al identificador (id) o su nivel lexicográfico es menor al nivel lexicográfico actual, se colocan las características del objeto a partir del tope de DES. Este valor, se guarda en el tope de INFO, a su vez, el tope de INFO pasa a ser la cabeza de la lista que vive en LI y es apuntada o direccionada desde LIG.

.-Las acciones necesarias para 'eliminar' los descriptores de un bloque son : restar uno de NIVEL, quitar el primer nodo de las listas que viven en LI y apuntar NIVELS[NIVEL] hacia afuera de DES.

Para ejemplificar el funcionamiento de la tabla de símbolos, supongamos se procesa el siguiente programa.

```
            INICIO
                REAL   x,yy
(1)               RUTINA r;
                    INICIO
(2)                           REAL      x;
                    FIN;
```

El estado de la tabla de simbolos, en los momentos (1) y (2) se muestra en la siguiente figura :

Tabla de simbolos en el momento 1.

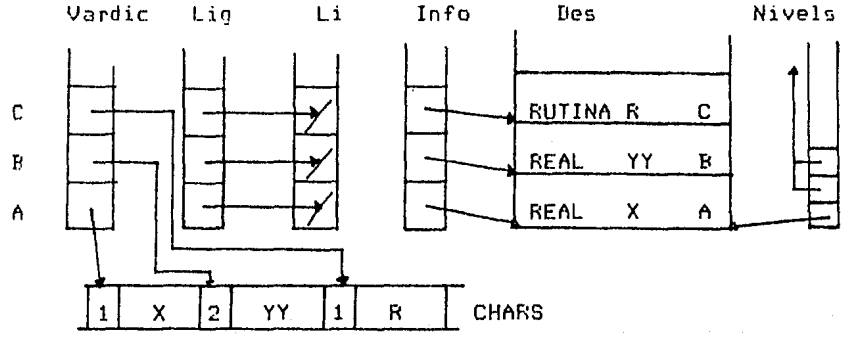


Tabla de simbolos en el momento 2.

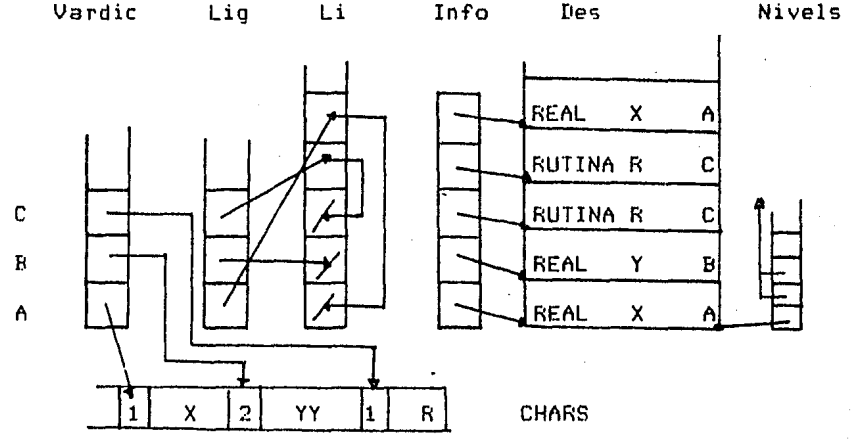


Diagrama 1.4

Descriptores.

Los descriptores de los objetos declarables en el lenguaje constan necesariamente de un descriptor primario, cuyos campos son:

1) Apuntador a la tabla de cabeza de listas (LIG) y al diccionario de átomos (VARDIC).
2) Tipo de variable.
3) Forma de variable
4) Localidad relativa al inicio del bloque, donde iniciará la representación del objeto.
5) Número de descriptores adicionales, posiblemente uno o más descriptores secundarios.

Los tipos de variable son :

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

La forma de una variable puede ser:

1.- Achivo para lectura
2.- Archivo para escritura
3.- Constánte
4.- Variable escalar o parámetro por valor
5.- Escalar como parámetro por referencia
7.- Gráfica
8.- Gráfica como parámetro por referencia
10.- Cuerpo 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

Para los descriptores adicionales, el primer campo es cero y la interpretación de los campos restantes depende del objeto

declarado. Además, el primer campo de los descriptores primarios se usa con un mismo fin, por lo tanto, es factible presentar los descriptores del lenguaje y omitir el primer campo de los descriptores.

Constante :

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

Escalar :

2) Real (1) o Texto(2)
3) Variable simple o parámetro por valor (4), parametro por referencia (5)
4) Localidad en memoria.Si es un parametro 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), si es parámetro por referencia (18). Arreglo de textos o columna con más de una dimensión (17). Si es parámetro por referencia (19).
4) Localidad en memoria. Si es parámetro por referencia, localidad donde vivira la dirección del arreglo.
5) Número de dimensiones.Si es pafametro por referencia contiene un uno

Segunda Parte.
Por cada dimensión.
2) Valor del límite inferior. Por razones de implantacion es siempre uno
3) Valor del límite superior
4 y 5) Cero

Si es parámetro por referencia, solo existe un descriptor adicional, donde el campo dos contiene el número de dimensiones y el resto del descriptor contiene ceros.

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) Cinco. 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

Para el campo CUERPO :

2) Real (1)
3) Cuerpo de Grafica (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

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 parametro 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 o columna 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: 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 las otras dos estructuras.
Campo NOMREN, igual que la estructura TABLA.

Campo CUERPO :

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

Dimensión del Histograma :

2) Numero de frecuencias
3, 4 y 5) Cero

Rutina :

2) Con tipo (12). Sin tipo (13)
3) Rutina (31)
4) Nombre interno
5) Número de parámetros .

Segunda Parte :

Por cada parámetro :
1) Apuntador a la tabla de cabezas de listas (LIG) y al diccionario de identificadores (VARDIC).
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 :

1) Archivo (10)
2) Archivo de lectura (1), de escritura (2)
3) Nombre interno
4) Apuntador al nombre externo.

Formato :

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

1.4.2 ADMINISTRACION DE MEMORIA.

El propósito de esta sección es describir cómo y dónde se representan las constantes, literales, variables escalares, arreglos, tablas e histogramas. El lugar donde se opera y representa a los objetos mencionados es un arreglo de reales llamado MEM.

Explícitamente el arreglo MEM está definido de las siguiente manera:

```
TYPE      TMULT=(RE,CAR,GRA)

          TIPOMEM = PARCKED RECORD
                    CASE TMULT OF
                        RE : ( R:REAL);
                        CAR : (C:ARRAY[0..5] OF CHAR);
                        GRA : (G:ARRAY[0..5] OF BYTE);
                    END;
VAR
          MEM: ARRAYE MINMEM..MAXMEM ] OF TIPOMEM;
```

donde MINMEM y MAXMEM son constantes

Así, es posible ver una localidad de memoria (48 bits) como 1 real, 6 caracteres o 12 puntos de una gráfica. MINMEM y MAXMEM son constantes que se determinan en función de la memoria principal de la computadora anfitriona.

Por ser el lenguaje estadístico un lenguaje con estructura de bloques, el compilador debe generar código capáz de 'desaparecer' las variables locales de un bloque cuando se desactiva éste, y 'aparecer' los objetos locales a una rutina cuando es activada, así mismo debe poder referenciar el objeto indicado cuando dos objetos comparten un identificador.

La memoria se administra en forma de stack, de manera que el área local a un bloque que se activa quedará a partir de la ultima localidad ocupada por el bloque que causó la activación. Por lo tanto, para conocer en ejecución la localidad exacta donde inicia la representacion de un objeto que viva en memoria, es necesario saber :

.-El nivel lexicográfico del objeto, conocido en compilación a través de la variable NIVEL.

.-Las primeras localidades ocupadas por los diferentes bloques activos. Estas direcciones se almacenan en el arreglo DISPLAY y se determina en ejecución.
Es importante recordar que un bloque de nivel lexicográfico L, no tendrá acceso a las variables declaradas en otro bloque con igual nivel lexicográfico.

.-El desplazamiento relativo al bloque. Este valor es conocido en compilación y almacenado en el descriptor del objeto.

entonces :

localidad en ejecución = DISPLAY[nivel lexicográfico] + desplazamiento relativo

Además, TOPEMEM apuntará al tope de la memoria, NIVEL indica el máximo nivel lexicográfico activo y MAXNIVEL es una constante igual al máximo nivel lexicográfico permitido.

TOPEMEM, DISPLAY, y NIVEL son variables que vivirán en el programa objeto.

Representación de variables

Una variable escalar real es representada por una celda de memoria.

Las constantes o literales tipo real no vivirán en memoria. Si se necesita su valor, en compilación se llama a la función VALOR que toma como argumento la localidad de CHARS donde se almacenaron los caracteres que representan a la constante o literal y regresa el valor de la constante o literal, para escribirlo en el programa objeto.

Una variable escalar tipo texto necesita para su representación de 11 celdas, diez para guardar hasta 60 caracteres y una para alamcenar la longitud de la cadena.

Una constante tipo texto se maneja casi igual que una variable, la diferencia estriba en que se inicializa y no se puede alterar su valor. La asignación es realizada por la rutina LECCAD encargada de copiar la cadena de CHARS a MEM.

Las literales tipo texto también vivirán en MEM, ya que durante la compilación al encontrar el uso de una literal, el código generado (llamado a LEC2CAD) leerá la literal de CHARS y la colocará en el tope de MEM.

Por razones de implantación el límite inferior de las dimensiones de un arreglo es uno y el límite superior debe ser mayor o igual a dos.

Conocer el número de dimensiones y las magnitudes de éstas en un arreglo en ejecución es de suma importancia, especialmente para los arreglos temporales.

Por ejemplo, supongamos las siguientes definiciones:

```
ARREGLO      A[1:10];
REAL            J;
```

Entonces, A[1_J] puede ser un arreglo o un escalar, dependiendo del valor de J, por lo que cada arreglo contará con un descriptor dinámico, como se explica a continuación.

La representación de un arreglo es la siguiente: en la primer localidad se almacena el número de dimensiones, si es un arreglo unidimensional 1 indicara un arreglo columna o texto y -1 arreglo renglón, si N es el número de dimensiones, las siguientes N celdas contienen el tamaño de cada dimensión, a continuación, se almacenan los elementos del arreglo por renglones.

La representación de Tablas, Gráficas o Histogramas se obtienen como agrupaciones de las estructuras mencionadas y la estructura cuerpo gráfica, siguiendo el orden dado por los descriptores.

Los campos TITULO, TITCOL, TITREN son variables escalares tipo texto. NOMREN y NOMCOL son arreglos unidimensionales de textos.

El cuerpo de un Histograma será un arreglo real unidimensional, el de una Tabla, es representado por un arreglo real bidimensional.

La descripción de la representación y operación del cuerpo de una gráfica se pospone al capítulo tres.

Los máximos y mínimos de abscisas y ordenadas de las gráficas son representadas como variables escalares reales.

1.4.3 COMPILACION DE EXPRESIONES

Siendo un objetivo importante del lenguaje, proporcionar herramientas que faciliten la operación y manejo de datos sobre todo numéricos, la compilación de expresiones es la piedra angular del compilador.

Es importante mencionar algunas características de la manera como se concibió la evaluación de expresiones:

-No puede actuar operador alguno sobre las variables tipo Tabla, Gráfica o Histograma. Por supuesto se puede operar con los componenetes de éstos.

-El operador suma o concatenación es el único que puede actuar sobre variables tipo texto o cuerpo gráfica.

-Si el resultado de una expresión es de tipo real, ésta se interpretará como lógica si es necesario. Por ejemplo en la proposición condicional. Una expresión será cierta si todos los elememtos de la estructura resultante son diferentes de cero.

-Los operadores suma y multiplicación son definidos como asociativos por la izquierda, por lo tanto, a +b + c es (a+b)+c.

-La asociatividad en la operación potencia debe hacerse explícita a**b**c será sintácticamente incorrecta.

-Otro error de sintáxis es ocasionado cuando existen 2 o más operaciones de comparación y estas no se encierran con paréntesis, por ejemplo :

a > b y b< c .

Se dirá que una variable lógica declarada en el programa objeto tiene el valor de Falso, sí en el código se le asigna el valor de FALSE. analogamente la variable tendrá el valor de Verdad sí en el código generado se le asigna el valor de TRUE.

Para realizar la compilación de expresiones se usa un stack de operadores y otro para los operandos.

El stack de operandos es definido de la siguiente manera:

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

VAR STOPNDO : ARRAYE 0..TSTOPNDO ] OF TOPNDO;
```

donde TSTOPNDO es una constante.

Es innecesario durante la ejecución distinguir un elemento de un arreglo real con un escalar real, por lo que se definieron 9 tipos (llamados tipos en ejecución), asociando un tipo por cada representación distinta de los objetos declarables, estos tipos son los que se graban al código generado.

Los tipos en ejecución son:

1.-Si el operando representa una constánte o literal real, el valor de la constante o literal se guarda en LOCR.

Todas las estructuras restantes necesitan para su descripción de LOC y NIV.Si el operando es un operando temporal , entonces LOC=-1 y NIV no se usa, en caso contrario, LOC será el desplazamiento relativo al bloque y ABS(NIV) el nivel lexicográfico.

Los tipos faltantes son:

2.- Varible escalar de tipo real.
3.- Variable escalar ó constánte ó literal de tipo texto.
4.- Arreglo real
5.- Arreglo de textos.
6.- Gráficas.
7.- Tablas.
8.- Histograma.
9.- Cuerpo gráfica.

La interpretación de NIV fue modificada en el presente trabajo; este aspecto es explicado en el capitulo cinco.

El stack de operadores se definió como sigue :

```
TYPE      TOPDOR : INTEGER;

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

donde TSTOPDOR es una constante.

Los operadores junto con su precedencia y clave se pueden leer en [E1].

La generación de código es sencilla, debido a que la gramática de expresiones es de precedencia de operadores, es decir los operadores de mayor precedencia aparecen más al fondo del árbol, evitandose la conversión a notación polaca.

Como ya se mencionó, el análisis semántico y la generación de código son procesos regidos por los llamados semánticos, y dependiendo de su función los llamados que compilan a una expresión los podemos dividir en:

-Los que meten un operador al stack de operadores.

-Los que meten un operando al stack de operandos.

-Aquellos que toman 1 o 2 operandos y un operador,para generar el código que realiza la operacion correspondiente, además de verificar que se cumplan las reglas semánticas posibles de validar en compilación.

-Un llamado especial(5027) cuya tarea es comprobar que es posible aplicar una expresión selectiva a un arreglo.

Evaluación de expresiones en ejecución

El código generado para evaluar expresiones, consiste en llamados a las rutinas OPERBIN y OPERUNA, cuya declaración es la siguiente:

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

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

Donde OPER es el nombre de alguna operación del lenguaje y TIPOPER incluye los nombres de todas las operaciones.

El operando sobre el que actuará OPERUNA queda determinado por los valores de las variables LC1, TP1 y NIVELUNO. LC1(localidad relativa al inicio del bloque) y TP1(tipo) se interpretan al igual que los elementos del stack de operandos en compilación, el nivel lexicográfico es pasado como parametro(si es necesario) a través de la variable global NIVELUNO.

En las operaciones binarias, el operando derecho es descrito por (LC2,TP2,NIVELDOS), NIVELDOS es una variable global.

Las rutinas OPERBIN y OPERUNA además de realizar las operaciones con los operandos, se encargan de realizar validaciones semánticas.

Debido a la definición del lenguaje, no es posible conocer en compilación el número de localidades ocupadas por un operando temporal.Por ejemplo si se declara:

```
ARREGLO A[1:10];
REAL I,J;
```

entonces A[I_J] es igual a A si I=1 y J=10 o A[I_J] es A[1] si i=j=1. Como se ve, es necesario conocer en ejecución el tipo de los operandos. La localidad donde vivirá un operando temporal se determina hasta ejecución, por ejemplo, el resultado de la expresión -A[I_J] se construirá despues de la última localidad ocupada por A[I_J] y como en compilación no sabemos su tamaño,no podemos saber donde vivirá -A[I_J], por lo que también es necesario conocer las localidades donde viven los operandos en ejecución. Para conocer las localidades y los tipos de los operandos durante la ejecución, se maneja un stack de operandos en ejecucion definido de la siguiente manera:

```
TYPE  TOPNDO = RECORD
                 CTIPO,CLOC : INTEGER;
                 END;

VAR STOPNDO :ARRAY[0..TAMSTOPNDO] OF TOPNDO;
```

Siendo TAMSTOPNDO una constante.

A continuación se presentan los estados por los que pasará el stack de operandos y la memoria al evaluar -(A+B)*(C+D), con A, B, C y D reales.

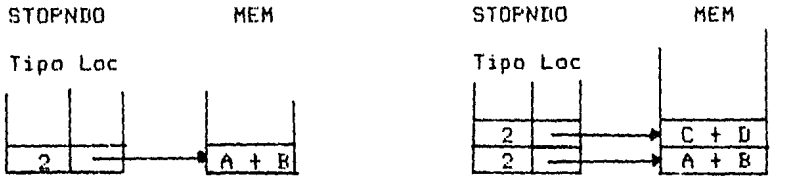


Diagrama 1.5

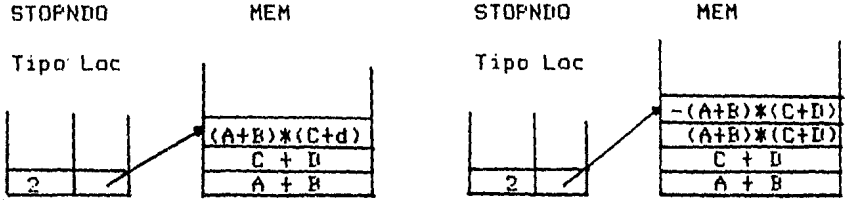


Diagrama 1.6

De esta manera, el manejo del stack es sencillo, pues los operandos temporales se construyen a partir del tope de la memoria y al realizarse una operación se sacan los descriptores de los operandos y se mete el descriptor del operando resultante.

Expresión Selectiva de Arreglo.

Es una expresión que actúa sobre un operando tipo arreglo, con el fin de seleccionar o sumar elementos de éste. La definición gramatical para la expresion selectiva de arreglo es:

```
<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
```

Los operadores permitidos en la expresión selectiva son:

```
Vacio          .-Se toman todos los elementos de la dimensión.
'.'(punto)     .-Se suman los elementos de la dimensión.
'_'(subrayado).-Marca un intervalo de elementos.
'&'            .- Une dos intervalos.
```

Si la expresión selectiva se aplica al arreglo destino de una asignación, indicará los elementos del arreglo donde se almacenará el resultado de evaluar la expresión.

No se permite el operador '.' (punto) cuando el operando es un arreglo de Textos o en la asignación a un arreglo de reales.

Arreglos explícitos.

Los arreglos explícitos son arreglos temporales cuyos elementos son expresiones. Se crean en ejecución antes de operar con ellos y desaparecen después de haber intervenido en la evaluación de la expresión.

La definición gramatical de un arreglo explícito es la siguiente:

```
<Arreglo Explicito> ::= '[[' <limites Exp> '|'
                            <Lista de Exp Rep> ']]' <5037>
<Lmites Exp> ::= <5036><Num o Cons><R7>
<R7> ::= ',' <Num o Cons>|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>
```

Al usar un arreglo explícito deben observarse las siguientes reglas:

-Un arreglo explícito no puede ser elemento de otro arreglo explícito.

-El arreglo explícito debe ser un arreglo de reales o textos.

1.4.4 PROPOSICION DE ASIGNACION

La proposición de asignación y la evaluación de expresiones forman una parte fundamental del lenguaje, pues son los elementos que permiten la manipulación y operación de datos.

A continuación, se presenta la definición gramatical de la proposición de asignación:

```
<Asig> ::= <5035> <Est ref> ':=' <5033><Exp><5034>
<Esr 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
```

El lado izquierdo puede consistir de una variable escalar, un arreglo, tabla, gráfica, histograma o algún campo de los tipos estructurados, además el lado izquierdo puede ser un arrreglo o el campo de un tipo estructurado que sea arreglo, operados por una expresión selectiva.

La expresión del lado derecho debe ser del mismo tipo y forma que la estructura del lado izquierdo. En el caso de ser arreglos deben tener el mismo número de dimensiones y si son cuerpos de gráfica, deben ser de igual tamaño.

1.4.5 ESTRUCTURAS DE CONTROL

En la tesis que antecedió a este trabajo se implantaron las proposiciones condicional y de trasferencia incondicional.

Los elementos manejados fueron:

- -Stack de proposiciones
- -Tabla de etiquetas
- -Interpretación lógica de una expresión real.
- -Llamados semánticos encargados de procesar a las proposiciones implantadas.

De los elementos citados, algunos serán usados y otros modificados o completados para lograr la implantación total de las proposiciones de control del lenguaje.

Por las razones anteriores y buscando una presentación consistente, se consideró conveniente presentar todas las proposiciones de control en el segundo capítulo.

Como ya se mencionó, en el desarrollo de este trabajo se trslado el compilador a las microcomputadoras PC. La descripción del traslado y las consecuencias de éste en el compilador son el contenido de las siguientes dos secciones.

1.5 TRASLADO DEL COMPILADOR

La traducción de Apple-Pascal a Turbo-Pascal consistió de:

-La transformación de los programas fuente del compilador de Apple-Pascal a Apple DOS V3.3 .

-Usar la tarjeta Quadlink para convertir archivos tipo texto de APPLE DOS V3.3 a archivos MS-DOS. El Instituto de Investigaciones Filológicas facilitó el uso de la tarjeta.

-Crear y ejecutar un pequeño programa encargado de indentar y filtrar los archivos.

-Modificar la declaración y uso de los archivos en los programas que forman el compilador.

-Turbo Pascal no posee el procedimiento:

PROCEDURE EXIT (p : procedure)
esto implicó simular algunos exit con varios GOTO, porque en Turbo Pascal un GOTO solo puede hacer referencia a las etiquetas locales al bloque.

En resumén el traslado de FranKlin a PC favoreció enormemente el desarrollo del proyecto, y se espera favorezca de igual manera a los posibles futuros usuarios del compilador.

1.6 MODIFICACIONES A LA IMPLANTACION DEL COMPILADOR

Se trasladó el compilador a las micromputadoras PC. Este hecho acarreo los siguientes beneficios:

-La memoria secundaria pasó de 280 K a 720 K, adicionalmente si la PC en la que se use el compilador tiene disco duro, puede hacerse uso de éste

-El almacenamiento de archivos en disco en cuanto a manejo y capacidad es bastante superior con respecto a la Franklin.

-La velocidad de procesamiento aumentó considerablemente, pues las microcomputadoras Franklin trabajan con un microprocesador de 8 bits (el 5052), mientras que las PC poseen un 8086 (16 bits) y es posible añadirles un procesador 8087.

-El editor de Turbo Pascal es más simple y potente que el editor de Apple Pascal.

-Turbo Pascal ocupa menos espacio que Apple Pascal y no genera código para un interprete, como el de Apple Pascal.

-La precisión en el tipo real aumentó. Anteriormente una variable tipo real se representaba con 4 bytes y ahora es representada con 6 bytes.

Por el contrario pasar el compilador a PC ocasionó algunos cambios en el compilador.

-Una variable tipo texto ocupa 11 celdas reales(una para la longitud y 10 donde almacenar caracteres) y en Franklin ocupaba 15 celdas reales para almacenar los caracteres, por lo que fue necesario modificar ligeramente algunas rutinas de biblioteca.

-Todos los archivos de textos, que el compilador tome como entrada, deben ser archivos en disco. Anteriormente se permitia que fueran leidos del teclado.

-No es posible ligar desde Turbo Pascal las rutinas de biblioteca, quedando la posibilidad de usar el Turbo Ligador.

ESTRUCTURAS DE CONTROL

2.1 ASPECTOS GENERALES

Las estructuras de control son importantes porque permiten modificar el flujo secuencial de ejecución.

Las proposiciones de control de flujo son parecidas a las definidas en los lenguajes ALGOL y PASCAL y pueden clasificarse en : iterativas, condicional, selectiva e incondicional. A continuación se señalan los aspectos generales en la compilación de las proposiciones de control para luego describir los aspectos particulares de cada proposición.

Con la finalidad de facilitar el análisis sintáctico de una proposición como la siguiente :

```
( 1 )   REPITE MIENTRAS I ESTO
        INICIO

( 2 )       I:= I+1;
        FIN;
```

es procesada como se explica a continuación : el reconocedor sintáctico reconocerá la proposición en dos partes, creando un árbol para (1) y otro para (2), es decir, al reconocer sintácticamente (1) no se verifica la existencia de la palabra reservada 'FIN', que indicaría sobre cuales proposiciones ejerce control la proposición 'REPITE MIENTRAS', este aspecto será revisado por el analizador semántico. Si la proposición usada fuera :

```
REPITE MIENTRAS I ESTO
    I:=I+1;
```

se reconoce en un solo árbol de derivación.

Cuando una proposición ejerce control sobre varias proposiciones, el conjunto de proposiciones afectado será delimitado por las palabra clave 'INICIO' y 'FIN'.

Desde la gramática, una proposición incondicional está definida por :

```
<PR INC> ::= <ASIG>|'INICIO'|<LLAMADO>|<P VER>|<P LECTURA>|
             <P ESCRITURA>
```

Así, las proposiciones incondicionales son : la proposición de asignación, un llamado a rutina, las proposiciones de lectura y escritura, la proposición de transferencia incondicional (P VER) y la palabra clave 'INICIO' que servirá para delimitar una lista de proposiciones. En la gramática no se define una proposicion compuesta, pero se entenderá a una proposición compuesta como una lista de proposiciones generales limitada por

las palabras clave 'INICIO' y 'FIN', de esta manera, la alternativa número dos de (PR INC) corresponde al 'INICIO' que formará parte de la proposición compuesta.

Por lo general, al aparecer una expresión en la gramática, esta es acompañada de dos llamados semanticos, el primero de ellos (5033) consiste en inicializar los stacks de operadores y operandos, además de llamar a la rutina PROCEXP, que es la encargada de generar propiamente el código que evaluará a la expresión. El segundo llamado (5034) es la señal de que termina el análisis semántico de una expresión (finaliza ejecución de PROCEXP). Esto es necesario porque es posible que PROCEXP se llame a si misma, por ejemplo en la expresión (X+Y)*(X+Y) cada expresión dentro de un paréntesis será procesada por un llamado a la rutina PROCEXP.

Asi, cuando se diga ' el llamado X genera código que evaluará a la expresión ', querrá decirse explicitamente : el llamado X usando la rutina PROCEXP (posiblemente invocada por el llamado 5033) generará el código necesario para evaluar a la expresión.

De igual manera, al decir ' se verifica el tipo de la expresión' deberá entenderse como el proceso formado por : sacar el elemento del tope del stack de operandos (en compilación) y conocer el posible tipo en ejecución de la expresión, para después, en caso de que el tipo no corresponda al esperado marcar un error.

Al no existir el tipo de dato lógico o booleano, algunas veces es necesario interpretar lógicamente a las expresiones de tipo real. La interpretación es realizada por la rutina EVALCOND(LOC,TIPO) que toma al operando representado por (LOC,TIPO) y a la variable CONDRES asignael valor de Verdad si (LOC,TIPO) representa un escalar real distinto de cero o un arreglo de reales con todos sus elementos distintos de cero, en caso contrario CONDRES será falsa. Si (LOC,TIPO) no representa a un número o arreglo real entonces CONDRES sera falsa .

Stack de proposiciones.

Para generar el código correspondiente a las proposiciones de control de flujo y detectar errores en el uso de estas, se utiliza un stack de proposiciones definido como se muestra en seguida :

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

con TMSTPROP una constante.

La palabra 'FIN' o ';' (punto y coma) pueden ser los delimitadores del ámbito de acción de la proposiciones de control de flujo. Saber a cuales proposiciones delimitan es importante

para la generación del código. Por lo tanto, todas las proposiciones de control de flujo tendrán una representación en el stack, con excepción de la proposición VER.

Los elementos del stack de proposiciones son interpretados como se explica a continuación:

```
 0.- No existe proposición en el stack.
 1.- Condicional sencilla.
        SI .. ENTONCES
 2.- Condicional compuesta.
        SI .. ENTONCES INICIO
 3.- Fin primer parte de condicional compuesta.
        SI .. ENTONCES INICIO .. FIN
 4.- Alternativa sencilla de condicional.
        SI .. ENTONCES .. SINO
 5.- Alternativa compuesta de condicional.
        SI .. ENTONCES .. SINO INICIO
 6.- Repite sencillo.
        REPITE ...
 7.- Repite compuesto.       REPITE INICIO
 8.- Mientras sencillo.
        REPITE MIENTRAS .. ESTO
 9.- Mientras compuesto.
        REPITE MIENTRAS .. ESTO INICIO
10.- Selectiva.
        LA .. DE INICIO
11.- Alternativa sencilla etiquetada de selectiva.
        ( etiquetas ) : y el tope del stack es 10
12.- Alternativa compuesta etiquetada de selectiva .
     ( etiquetas ) : INICIO y el tope del stack es 10
13.- Fin primera parte de selectiva.
        LA .. DE INICIO .. FIN
14.- Alternativa sencilla no etiquetada de selectiva.
        LA .. DE INICIO .. FIN SINO
15.- Alternativa compuesta no etiquetada de selectiva.
        LA .. DE INICIO .. FIN SINO INICIO
16.- Repite con sencillo.
        REPITE CON .. ESTO
17.- Repite con compuesto.
        REPITE CON .. ESTO INICIO
18.- Rutina.
```

El stack de proposiciones deberá quedar vacío al finalizar el análisis del programa fuente. De no suceder esto, se señala un error.

2.2 PROPOSICION ITERATIVA

La definición gramatical de la proposición iterativa es :

```
< ITER >::= 'REPITE'<5050><CUERPO ITER>
< CUERPO ITER >::= 'INICIO'|<P INC><HAS><TERMINA>|
  'CON'<IDENTIF>':=' <5033><EXP><5034>'SUMANDO'
  <5033><EXP><5034>'HASTA'<5033><EXP><5034>'ESTO'<P INC>|
  'MIENTRAS'<5033><EXP><5034>'ESTO'<P INC>
```

dependiendo de la alternativa tomada en <CUERPO ITER> ,la proposicion iterativa corresponderá al FOR de BASIC o al WHILE o REPEAT de PASCAL.

Proposición Repite.

Las producciones gramaticales que completan la definición de la proposición REPITE son :

```
< TERMINA >::= 'FIN'<5045><RFI><TERMINA>|e
< HAS >::= 'HASTA'<5051><5033><EXP><5034>
```

Si la alternativa tomada en (CUERPO ITER) es 1 o 2 ,el programa fuente debe contener una proposició de la siguiente forma :

```
REPITE
   Proposición incondicional.
HASTA expr;
```

donde expr es una expresión real y la proposición incondicional puede ser una proposición compuesta. El significado de la proposición REPITE es el siguiente : se ejecuta la proposición incondicional y se evalúa a la expresión (que deberá de ser real). La iteración termina cuando la expresión resulta ser verdadera.

Simbólicamente el código generado es :

```
REPEAT
   { Código para la proposición incondicional     }
   { Código que evalúa e interpreta a la expresión }
UNTIL CONDRES;
```

es necesario recordar que CONDRES es una variable booleana declarada en el programa objeto, en la que la rutina EVALCOND depositará el resultado de interpretar lógicamente a la expresión.

De esta manera, si la rutina DECREP (llamado 5050) detecta que la alternativa de (CUERPO ITER) es 1 o 2, genera a la palabra REPEAT y mete al stack de proposiciones un Repite sencillo (6) o Repite compuesto (7), dependiendo de, si la proposición incondicional es una proposición compuesta, es decir, la alternativa tomada en (CUERPO ITER) es 1.

Si la rutina CCOMP (Cierra proposiciones compuestas llamado 5045) encuentra en el tope del stack de proposiciones un Repite compuesto (7), lo cambiará por un Repite sencillo (6) para indicar que apareció el 'FIN' que cierra al 'INICIO', que señala el principio de la proposición compuesta.

Luego, al ejecutarse la rutina FINREP (llamado 5051), ésta debe encontrar necesariamente un Repite sencillo (6) en el tope del stack. De no ser así, se señala un error, además la rutina FINREP (llamado 5051) genera código para evaluar a la expresión y verifica que el tipo de la expresión sea real.

Proposición Repite Mientras.

La proposición iterativa funciona como un WHILE de PASCAL cuando la alternativa tomada en (CUERPO ITER) es 4, suponiendo que expr es una expresión real, simbólicamente la proposición iterativa es :

```
REPITE MIENTRAS  expr  ESTO
   { PROPOSICION INCONDICIONAL };
```

La proposición REPITE MIENTRAS se interpreta asi : se evalúa a la expresión e interpreta como booleana, si es verdadera se ejecuta a la proposición incondicional, para después evaluar e interpretar nuevamente a la expresión, la iteración termina cuando la expresión es falsa.

Simbólicamente el código generado será :

```
REPEAT
   { Código para evaluar e interpretar a la expresión }
   IF CONDRES THEN BEGIN
      { Código de la proposición incondicional }
   CONDRES := TRUE;
   END;
UNTIL NOT CONDRES;
```

por lo tanto, si la alternativa de (CUERPO ITER) es 4, la rutina REPITE (llamado 5050) genera : a la palabra REPEAT, el código que evaluara e interpretara a la expresión y la linea 'IF CONDRES THEN BEGIN', además valida el tipo de la expresión y dependiendo de la alternativa tomada en (PR INC), meterá al stack de proposiciones un Mientras compuesto (9) si la alternativa corresponde al 'INICIO', si no, meterá un Mientras sencillo (8).

Las últimas tres lineas son generadas por las rutinas CCOMP o CSIMPLE (llamados 5045 y 5047), contenidos en las siguientes producciones :

```
< TERMINA > ::= 'FIN'<5045><RFI><TERMINA>|e
< P > ::= ...|<5042><L ETIQ><P GRAL><5047>|..
```

estas líneas son generadas por la rutina CCOMP (llamado 5045) si el tope del stcaK de proposiciones es Mientras compuesto (9) o por la rutina CSIMPLE (llamado 5047) si el tope del stacK de proposiciones es Mientras sencillo (8), cualquiera de los dos llamados que actue, quitará el elemento del tope del stacK de proposiciones.

Proposición Repite Con.

Esta proposición permite ejecutar una proposición incondicional, dependiendo del valor que tenga una variable escalar real (llamada variable de control). Los valores que toma la variable de control formarán una progresión aritmética, la variable de control debe ser una variable escalar real que no sea parámetro por referencia..

La definición gramatical de la proposición REPITE CON es :

```
'CON'<IDENTIF> ':=' <5033><EXP><5034>'SUMANDO'<5033><EXP><5034>
     'HASTA'<5033><EXP><5034>'ESTO'<P INC>
```

Su significado es idéntico al FOR de BASIC, es decir en la inicialización del ciclo son evaluadas las tres expresiones (cuyos resultados deben ser escalares reales), la primer expresión es almacenada en la variable de control y las restantes en localidades temporales.

Si el valor de la segunda expresión es positivo, la iteración del ciclo consiste de : si el valor de la variable de control es menor o igual al valor de la tercera expresión, se ejecuta la proposicion incondicional (representada por <P INC>), e incrementa el valor de la variable de control en una cantidad igual al valor de la segunda expresión. Cuando el valor de la segunda expresión no es positivo, la condición para ejecutar la proposición incondicional es : si el valor de la variable de control es mayor o igual al valor de la tercer expresión.

Supóngase que X e I son variables escalares reales, expr1, expr2 y expr3 son expresiones cuyo resultado es un escalar real y considérese la siguiente proposición :

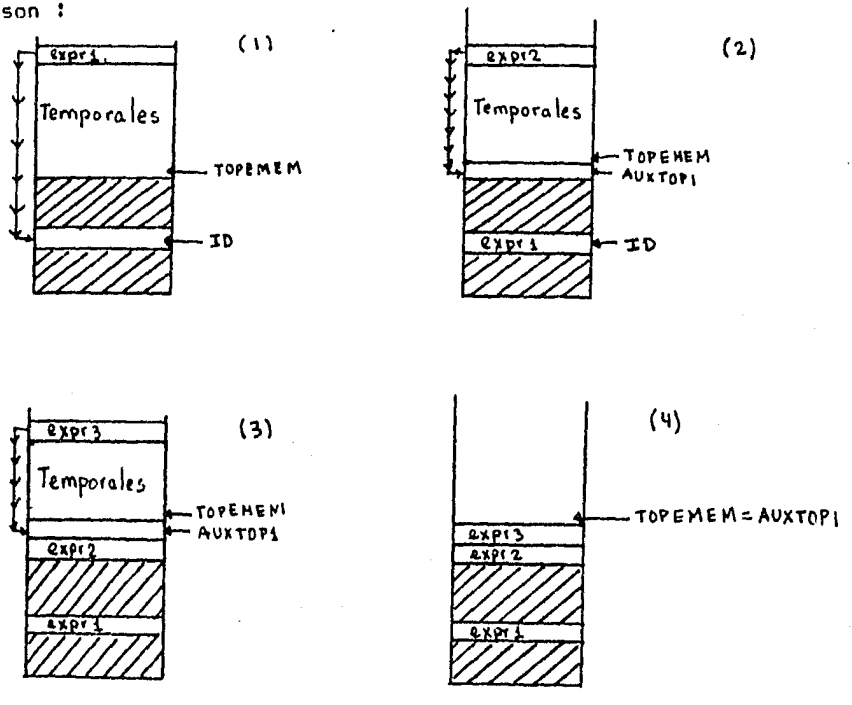
```
REPITE CON I:= expr1 SUMANDO expr2 HASTA expr3 ESTO
    X:=X+1;
```

entonces, si ID es una función que regresa la localidad en ejecución donde vive la variable de control, simbólicamente el código generado es :

```
        { Código que evalúa expr1 }
        TOPEMEM:= AUXTOP1;
1)      MEM[ ID ]:= valor de expr1
        { Código que evalúa a expr2 }
        TOPEMEM:= AUXTOP1 + 1;
```

```
2)      MEMC TOPEMEM - 1 ]:= valor de expr2
        { Código que evalúa expr3 }
        TOPEMEM:= AUXTOP1+1;
3)      MEMC TOPEMEM - 1 ]:= valor de expr3
        AUXTOP1 := TOPEMEM;
4)      WHILE SALDEFOR(ID,AUXTOP1) DO BEGIN
            { Código para X:= X + 1 }
5)          MEMC ID ] := MEMC ID ] + MEMC TOPEMEM - 2 ];
        END;
        TOPEMEM := TOPEMEM -2 ;
```

Gráficamente los estados por los que pasa el stack de memoria son :



Las figuras (1),(2) y (3) corresponden a los momentos (1), (2) y (3) en la ejecución del programa objeto. Es conveniente recordar que la variable AUXTOP1 sirve para recuperar el espacio ocupado por los temporales en la evaluación de una expresión.

En el momento (4) ya se han almacenado los valores de las expresiones e inicia la iteración.

La función SALDEFOR toma los valores de la variable de control y de las expresiones dos y tres, y decidira si el ciclo debe continuar.

En el momento 5 (figura 4) se incrementa el valor de la variable de control, cuando termina la iteración las dos localidades temporales ocupadas por las expresiones dos y tres son recuperadas.

Cuando la rutina REPITE (llamado 5050) encuentra que la alternativa de (CUERPO ITER) fue 3, verificara que el identificador usado este declarado y corresponda a una variable escalar real, generará código para evaluar a las expresiones, revisar en ejecución que los resultados de las expresiones sean de tipo escalar real y asigne las expresiones a las localidades temporales y a la variable de control.

Como en el punto (5), es necesario conocer el desplazamiento y nivel lexicográfico de la variable de control. En el stack de proposiciones se guarda un apuntador a la tabla de descriptores (DES), señalando el lugar donde inicia el descriptor de la varia ble de control. En seguida se mete al stack de proposiciones un Repite con sencillo (16) o Repite con compuesto (17), dependiendo de, si la alternativa tomada en (PR INC) corresponde a la palabra clave 'INICIO'.

La rutina CCOMP (llamado 5045) actúa si el tope del stack de proposiciones Repite con compuesto (17), pues ha aparecido la palabra 'FIN' que 'cierra ' a la proposición REPITE CON, y las acciones realizadas son : sacar el tope del stack, volver a sacar el tope del stack para obtener el nivel lexicográfico y el desplazamiento relativo al bloque de la variable de control y generar el código que actualice a la variable de control.

Idénticas acciones son realizadas por larutina CSIMPLE (llamado 5047), si el tope del stack de proposiciones es Repite con sencillo (17).

2.3 PROPOSICION SELECTIVA

Esta proposición es casi equivalente a la proposición CASE de PASCAL, pues permite seleccionar una o ninguna de varias proposiciones alternativas dependiendo del valor de una expresión.

En seguida, se muestran las producciones de la gramática en las que aparecen los llamados semánticos que intervienen en la compilación de la proposición selectiva :

```
< SELEC > ::= 'LA'<5052><5033><EXP><5034>'DE''INICIO'
< P > ::= ...|<5042><L ETIQ><P GRAL><5047>| ...
< L ETIQ > ::= <NUMERO> ':' <L ETIQ>|e
< TERMINA > ::= 'FIN'<5045><RFI><TERMINA>|e
< RFI > ::= <HAS><RC1>
< RC1 > ::= 'SINO'<P GRAL>
```

La proposición consta de : un encabezado definido por <SELEC>, un número arbitrario de proposiciones generales identificadas por listas no vacias de etiquetas numericas, la palabra 'FIN' que indicará que terminan la proposiciones alternativas etiquetadas y opcionalmente, la palabra 'SINO' seguida de una proposición general.

El significado de la proposición es el siguiente : se evalúa la expresión cuyo resultado debe ser un escalar real, se redondea y se busca la primera lista de etiquetas donde aparece.Si apareció en una lista, se ejecuta la proposición general asociada a la lista, en caso contrario, de aparecer la palabra clave 'SINO' se ejecuta la proposición general que le sigue,en cualquiera de los dos casos despúes de ejecutar a la proposición general adecuada, se pasa a ejecutar la proposición que sigue a la proposición selectiva. Si el resultado redondeado de la expresión no aparece en alguna lista y no existe la palabra 'SINO', ninguna alternativa se ejecutará.

Con el fin de ejemplificar, consideremos la siguiente proposición :

```
LA expr1 DE INICIO
  11:12 : INICIO     FIN;
     13 : X:=0;
  FIN SINO INICIO FIN;
```

donde 11,12 y 13 son literales numéricas, expr1 es una expresión real escalar y X es una variable escalar real.

Simbólicamente el código generado es :

```
{ codigo que evalúa y verifica el tipo de la expresión }
VARCASE := resultado redondeado de la expresión
ALTCASE := FALSE;
IF NOT ALTCASE THEN
   IF ( (VARCASE = 11 ) OR ( VARCASE = 12)  ) THEN BEGIN
      ALTCASE:=TRUE;
   END;
IF NOT ALTCASE THEN
   IF ( (VARCASE=13) ) THEN BEGIN
      { Código para X := 0 }
      ALTCASE := TRUE;
   END;
IF NOT ALTCASE THEN BEGIN
END;
```

siendo ALTCASE una variable booleana y VARCASE una variable entera, ambas declaradas en el programa objeto. Asignando a ALTCASE el valor de Verdad, se indica que ya se ejecuto una alternativa de la proposición selectiva, de esta manera, no se ejecutará alguna otra.

No se usó la proposicion CASE de PASCAL por dos razones:

- En la definición original de PASCAL no se contempla a la extensión OTHERWISE (Pascal Burroughs) o ELSE (Turbo - Pascal) que es equivalente a la parte SINO ..., usar una proposición CASE que no tenga a la extension ELSE, implicaría realizar el análisis semántico en dos pasadas, para evitar la dependencia del PASCAL utilizado se decidió generar el código como se explicó.

- Si se repitieran etiquetas en una proposición CASE , PASCAL señalaría un error de semántica, para detectar este error es necesario conocer en todo momento, las etiquetas pertenecientes a las proposiciones selectivas cuya compilación no ha terminado.

En seguida, se describe el papel de los llamados semánticos que dirigen la compilación de la proposición selectiva.

En primer lugar, la rutina DECCASE (llamado 5052) mete una Prop. selectiva (10) y genera código que : evalúe a la expresión, verifique su tipo en ejecución, asigne el valor redondeado de la expresión a la variable VARCASE, además de generar a la linea 'ALTCASE := FALSE; ' y revisar el posible tipo en ejecución.

La rutina DECET (llamado 5042) al encontrar una Prop. selectiva (10) en el tope del stack de proposiciones y si el primer átomo de la frase no es la palabra 'FIN', es la señal de que la lista de etiquetas no debe ser vacía y esta pertenecerá a un proposición selectiva.

Toda etiqueta encontrada por el compilador en el programa fuente será redondeada.

Al encontrar la primer etiqueta (e1) se generará :

```
IF NOT ALTCASE THEN
   IF ( (VARCASE = e1 )
```

para las siguiente etiquetas (e), se generará :

```
OR ( VARCASE = e )
```

al terminar la lista de etiquetas se grabará al código :

```
) THEN BEGIN
```

y se meterá al stack de proposiciones una Alternativa sencilla etiquetada de selectiva (11) o una Alternativa compuesta etiquetada de selectiva (12), dependiendo de, si (P GRAL) deriva o no en la palabra reservada 'INICIO'.

Si la lista de etiquetas es no vacia y el tope del stack es distinto de Prop. selectiva (10), los elementos de la lista de etiquetas se tratarán como posibles blancos de alguna proposición VER; cuando el tope del stack es una Prop. selectiva (10) y el primer átomo no es la palabra 'FIN' y la lista de etiquetas es vacia, se señala un error.

La rutina CCOMP (llamado 5045) generará :

```
ALTCASE:=TRUE;
END;
```

y sacará el tope del stack de proposiciones si este es una Alternativa compuesta etiquetada de selectiva (12). La rutina CSIMPLE realizará identicas acciones si el tope del stack es Alternativa sencilla etiquetada de selectiva (11).

Nuevamente la rutina CCOMP (llamado 5045) actua si el tope del stack de proposiciones es Prop. selectiva (10), pues es la señal de que terminarón las alternativas etiquetadas de la proposición selectiva y las acciones realizadas son : sacar el elemento del tope del stack y en caso de existir la proposición alternativa no etiquetada (SINO..) llamar a la rutina SECCON y meter al stack de de proposiciones un Fin primera parte de selectiva (13).

La rutina SECCON sirve para procesar a la alternativa no etiquetada de la proposición selectiva y a la segunda alternativa de la proposición condicional.

Si el tope del stack de proposiciones es Fin primera parte de selectiva (13), entonces la rutina SECCON generará a la linea

```
IF NOT ALTCASE THEN BEGIN
```

sacará el elemento del tope del stack (Fin primera parte de selectiva) y meterá una Alternativa simple no etiquetada de selectiva (14) o una Alternativa compuesta no etiquetada de selectiva (15), dependiendo de si la proposicion general que sigue al 'SINO' es una proposición compuesta.

Finalmente, si existió la alternativa no etiquetada de la proposición selectiva, la rutina CCOMP sacará el elemento del tope del stack de proposiciones y generará un 'END;' cuando el tope del stack sea una Alt. compuesta de no etiquetada de selectiva (15). La rutina CSIMPLE (llamado 5047) realizara las mismas tareas si el tope del stack es una Alternativa sencilla no etiquetada de selectiva (14).

2.4 PROPOSICION CONDICIONAL

Esta proposición tiene la misma estructura que el IF de PASCAL, está formada por : la palabra 'SI', seguida de una expresión real, la palabra 'ENTONCES', una proposición incondicional y opcionalmente se añade la palabra 'SINO' y una proposición general.

La expresion mencionada debe ser de tipo real pero se interpretará lógicamente, si la expresión es cierta se ejecutará la proposición incondicional, cuando la expresión sea falsa y existe la palabra 'SINO' se ejecutará la proposición general que sigue al 'SINO'.

Las producciones que intervienen en la definición gramatical de la proposición son las siguientes :

```
< COND >::= < P SI ><RC1>
< P SI  >::= 'SI'<5033><EXP><5034>'ENTONCES'<5044><P INC>
< RC1 > ::= 'SINO'<5046><P GRAL>|e
```

Con el fin de ilustrar la manera en que es generado el código, supónganse dos proposiciones SI, con sus respectivos códigos generados (considerando que expr es una expresion de tipo real)

PROPOSICION	CODIGO
SI expr ENTONCES;	{ Código que evalúa e interpreta e expr} IF CONDRES THEN BEGIN END;
SI expr ENTONCES SINO INICIO FIN;	{ Código que evalúa e interpreta a expr} IF CONDRES THEN BEGIN END ELSE BEGIN END;

En los siguientes parrafos, se explica la manera en que se compila la proposición condicional.

Los llamados 5033 y 5034 se encargan de generar el código que evaluará la expresión

La rutina CONDICIONAL (llamado 5044) verifica que el tipo de la expresión sea real, y genera la linea :

```
IF CONDRES THEN BEGIN
```

y un llamado a la rutina EVALCOND, que interpretará en ejecución el resultado de la expresión. Además la rutina CONDICIONAL, meterá al stack de proposiciones una Condicional compuesta (2), si la proposición incondicional deriva en la palabra 'INICIO' o una Condicional sencilla (1) en otro caso.

Otras dos rutinas que intervienen en la compilacion de la proposición condicional son la rutina CCOMP y la rutina CSIMPLE.

Cuando la rutina CSIMPLE encuentra en el tope del stack de proposiciones una Condicional compuesta (2) y la alternativa usada en (RC1) es Vacío (2) , generará un 'END;' y sacará el elemento del tope del stack ,Si la alternativa en (RC1) fue Condicional sencilla (1) , se saca el elemento del tope del stack, se mete un Fin primera parte de condicional compuesta (3) y se llama a la rutina SECCON.

Si el tope del stack de proposiciones es Condicional sencilla (1), la rutina CSIMPLE (llamado 5047) generará un 'END;' y sacara el elemento del tope del stack.La rutina CSIMPLE (llamado 5047) no puede invocar a la rutina SECCON porque el llamado 5047 se encuentra justo antes del punto y coma ';', que es el fin de la proposicion. La rutina SECCON puede ser llamada por CCOMP o por el programa principal (considerese el ejemplo 2).

Las funciones a realizar por la rutina SECCON son :

- Si el tope del stack no es :
  1.- Condicional sencillo.
  3.- Fin primera parte condicional compuesta.
  13.- Fin primera parte de selectiva.
  se señala un error.

- Si el tope del stack es condicional sencilla (1) o Fin primera parte condicional compuesta (3), sacará el elemento del tope del stack y generara a la linea :

  END ELSE BEGIN

  por último, si la proposicion general que sigue al 'SINO' es una proposición compuesta, meterá al stack una Alternativa compuesta de condicional (5) o una alternativa sencilla de condicional(4) cuando la proposición general (P GRAL) no deriva en la palabra 'INICIO'.

- Si el tope del stack es Fin primera parte de selectiva(13), generará a la linea :

  IF NOT ALTCASE THEN BEGIN

  además, cuando la proposición general que sigue al 'SINO' sea una proposición compuesta, mete al stack de proposiciones una Alternativa compuesta no etiquetada de selectiva (15), en caso contrario, se mete una Alternativa sencilla no etiquetada de selectiva (14).

Finalmente, la rutina CCOMP vuelve a intervenir si encuentra en el tope del stack una Alternativa compuesta de condicional (5), en cuyo caso generará un 'END;' y sacará el tope del stack de proposiciones. Idénticas tareas son realizadas por la rutina

CSIMPLE (llamado 5047) si el tope del stack es una Alternativa sencilla de condicional (4).

Como pudo verse, las rutinas CSIMPLE y CCCOMP juegan un papel importante en la compilación de las proposiones de control de flujo, por lo que se explicará con detalle su funcionamiento.

Si VSTPROP es una función que entrega el valor del tope del stack de proposiciones, la rutina CSIMPLE (llamado 5047) invocada al encontrar un punto y coma ";", tiene la siguiente estructura :

```
V := VSTPROP;
WHILE V IN [ 1,4,6,8,11,14,16 ] DO BEGIN
   CASE V OF
       1 : FINIF1 ; { cierra condicional sencilla }
       2 : FINIF2 ; { cierra condicional compuesta }
        .
   END; { DEL CASE }
   V:= VSTPROP;
END; { DEL WHILE }
```

donde para cada proposición en el stack existe una alternativa del CASE con su respectiva rutina.

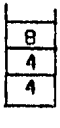
Como se sabe, un punto y coma puede delimitar el conjunto de instrucciones sobre el que actúa una proposición de control de flujo, por ejemplo :

```
SI expr ENTONCES
   SINO expr ENTONCES
      SINO REPITE MIENTRAS expr ESTO;
```

y suponiendo que expr es una expresión de tipo real, simbólicamente el código generado sera :

```
{ Código que evalúa e interpreta a la expresión }
IF CONDRES THEN BEGIN
END ELSE BEGIN
    { Código que evalúa e interpreta a la expresión }
    IF CONDRES THEN BEGIN
       REPEAT
          { Codigo que evalúa e interpreta a la expresión}
          IF CONDRES THEN BEGIN
1)           CONDRES:= TRUE;
          END;
       UNTIL NOT CONDRES;
2)  END;
  END;
```

STACK DE PROPOSICIONES CUANDO SE INVOCA A LA RUTINA CSIMPLE.



la rutina CSIMPLE actúa al aparecer el punto y coma, así al encontrar el Mientras sencillo (8) en el tope del stack generará las lineas señaladas por (1), sacara el tope del stack, para despues cada vez que saque una Alternativa sencilla de condicional (4) generar un 'END;'.

La rutina CCOMP (llamado 5045) se encarga de 'cerrar' una proposición compuesta, a diferencia de CSIMPLE que puede cerrar una o varias proposiciones simples.

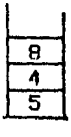
La estructura de la rutina CCOMP es la siguiente :

```
CSIMPLE;
V:= VSTPROP;
IF V IN [ 0,2,5,7,9,10,12,15,17,18] THEN
   CASE V OF
      0 : BFINSEMANTICA;
      2 : FINIF1; { cierra if compuesto }
      5 : FINIF2; { fin de alternativa de condicional }
       .
    END; { del case }
 ELSE   ERROR(' Error de semántica en el uso de
                proposiciones de control de flujo ');
```

La primer acción a realizar es un llamado a la rutina CSIMPLE, pues un 'FIN' puede significar el fin de una o varias proposiciones simples, por ejemplo supóngase I es un escalar real y considérese la siguiente proposición :

```
SI I ENTONCES
   SINO INICIO
           SI I ENTONCES
           SINO REPITE MIENTRAS I ESTO
       FIN;
```

El aspecto del stack de proposiciones, con anterioridad a la aparición de la palabra 'FIN' es el siguiente :



La palabra 'FIN' es la señal para generar el código que 'cierre' a la proposición REPITE MIENTRAS (Mientras sencillo.-8), la alternativa sencilla de condicional (4) y la alternativa compuesta de condicional (5).

Después de actuar CSIMPLE, en función del tope del stack se llama a la rutina adecuada, si el stack de proposiciones es vacío se asume que es el fin del programa fuente y se llama a la rutina BFINSEMANTICA, encargada de finalizar el análisis semántico.

2.5 PROPOSICION VER

Esta proposición permite alterar incondicionalmente el flujo natural de ejecución. Ejemplo del uso de esta proposición es el siguiente segmento de un programa fuente.

```
INICIO
    REAL I;
3: I:= I+1;
   VER 3;
```

Al ejecutarse la proposición VER, la ejecución del programa debe continuar a partir del punto donde se encuentra la etiqueta a la que se hace referencia. La proposición VER puede hacer referencia a cualquier etiqueta, siempre y cuando se encuentre en el mismo bloque de la proposición VER y no implique "entrar" a alguna de las siguientes proposiciones : iterativa, selectiva y condicional. Al igual que FORTRAN las etiquetas son numéricas y no se declaran.

La proposición GOTO de PASCAL difiere de la proposición VER en dos aspectos, las etiquetas deben declararse y algunas versiones de PASCAL permiten hacer referencias a etiquetas externas al bloque de la proposición VER.

Para no realizar dos pasadas sobre el programa fuente, una para identificar a las etiquetas y otra para declararlas, se decidío declarar un conjunto fijo de etiquetas por cada bloque y asociar a cada etiqueta del program fuente, una etiqueta del programa objeto, explícitamente se declaran las etiquetas : 1,2,3 .. MAXET, donde MAXET es una constante.

Los llamados semanticos que procesan a la proposición VER, se encuentran en las producciones :

```
< P > ::= <DECL>|<5042><L ETIQ><P GRAL><5047>|e
< L ETIQ >::=<NUMERO>":"<L ETIQ>|e
< P VER >::= "VER"<5042><NUMERO>
```

Para fines de compilación se considera que una etiqueta es declarada cuando aparece como posible blanco de un proposicion VER y es usada cundo la etiqueta es referenciada por una proposición VER.

Una etiqueta tiene en cualquier momento uno y sólo uno de los siguiente estados, representados por un valor, y ejemplificados con las siguientes proposiciones :

```
VER 8;
SI I ENTONCES INICIO 7: I:=0;   FIN;
```

-- Etiqueta declarada cuyo uso no es permitido (VALOR = 2), en el ejemplo, el uso de la etiqueta 7 queda vedado despúes de aparecer la palabra "FIN".

-- Etiqueta declarada cuyo uso es permitido (VALOR = 1). En el ejemplo, la etiqueta 7 tendrá un valor igual a 1, desde su declaración hasta la palabra 'FIN'.

-- Etiqueta usada cuya declaración es permitida (VALOR = 0).

-- Etiqueta usada cuya declaración no es permitida (VALOR<0). En el mismo ejemplo, la etiqueta 8 tendrá un valor de cero, hasta el momento en que aparece la palabra SI, durante la proposición SI .. su valor será de -1, hasta encontrar a la palabra FIN, para que la etiqueta 8 recupere su valor de cero.

La estructura de datos con la que se maneja a las etiquetas es una tabla definida por :

```
VAR TGOTO : ARRAYE 1..MAXET ] OF RECORD
                                  NOM,VAL :INTEGER;
                                END;
```

Las características que tiene una etiqueta son : su nombre externo (campo NOM), un nombre interno igual al índice del elemento de TGOTO donde vive el nombre externo y su valor (campo VAL).

Al aparecer por vez primera una etiqueta, se debe guardar el nombre externo de ésta en TGOTO e inicializar el valor de la etiqueta. La actualización del valor de las etiquetas es realizado por las rutinas CANCET1, CANCET2, DECET y BRINCA.

La rutina CANCET1, actúa cuando en el programa fuente aparece alguna de las siguientes palabras 'REPITE', 'SI', 'SINO' o al iniciar la compilación de una alternativa de la proposición selectiva, CANCET1 disminuye en uno el valor de todas las etiquetas que tienen un valor menor o igual que cero.

La rutina CANCET2, es invocada al terminar la compilación de la proposición iterativa o una proposición alternativa de las proposiciones condicional o selectiva, CANCET2 hace el campo VAL igual a 2 para todas aquellas etiquetas que tenian un valor igual a uno,así para las etiquetas declaradas en el ámbito de la proposición (iterativa o alternativa de condicional o selectiva) pasan a ser inaccesibles desde cualquier otro lugar del programa fuente, además para aquellas etiquetas con un valor menor a cero, se incrementa su valor en uno.

El llamado a la rutina DECET (llamado 5042) aparece junto con la lista de etiquetas, si el tope del stack no es Proposicion selectiva (10) y L es el nombre interno de una etiqueta, DECET ejecuta alguna de las acciones dependiendo del valor de la etiqueta .

Valor	Acción
1,2	.- Es señalado un error porque la etiqueta ya fue declarada.
0	.- Se genera 'L:; ' y se hace el valor de la etiqueta igual a uno.
< 0	.- Se señala un error y la etiqueta es almacenada con un valor igual a uno.

Finalmente, si la etiqueta no ha sida declarada ni usada, se almacena con un valor igual a uno y se genera 'L:;'.

La rutina BRINCA (llamado 5043) busca si se ha declarado o usado la etiqueta representada por (NUMERO) (en la definición de <P VER>), si la etiqueta ya existe con un valor igual a 2 es marcado un error, cuando la etiqueta no ha aparecido como usada o declarada la etiqueta se almacena con un valor de cero. Siempre que el valor de la etiqueta no sea dos se genera la línea :

```
GOTO I;
```

donde I es el nombre interno de la etiqueta.

2.6 MODIFICACIONES A LA IMPLANTACION DEL COMPILADOR.

Las modificaciones o adiciones hechas al compilador y presentadas en este capítulo son :

-- Se completo el manejo del stack de proposiciones.

-- Se añadieron las rutinas:

- REPITE (llamado semántico 5050)
- FINREP (llamado semántico 5051)
- DECCASE (llamado semántico 5052)

-- Se reescribieron las rutinas.

- DECET (llamado semántico 5042)
- BRINCA (llamado semántico 5043)
- SECCON (llamado semántico 5046)
- CSIMPLE (llamado semántico 5047)
- CCCOMP (llamado semántico 5045)

-- Fue modificada la gramática, el llamado 5042 estaba contenido en :

<L ETIQ> ::= <5042><NUMERO> ':' <L ETIQ>|e

ahora se encuentra en :

<P>::= <DECL>|<5042><L ETIQ><P GRAL><5047>|e

-- Se modificó totalmente el manejo e interpretación de la tabla de etiquetas. Ahora se detectan todos los errores en el uso de la proposición VER.

-- La proposición selectiva fue ligeramente modificada de su diseño original, pues se permite que una etiqueta aparezca en dos listas de etiquetas.

ALGUNOS ELEMENTOS DE LA EVALUACION DE EXPRESIONES

Para completar la implantación de la evaluación de expresiones, se adicionaron los siguientes elementos :

--- Suma de cuerpos gráfica.
--- Inversa de matrices.
--- Determinante de matrices.
--- Función logaritmo.

En cada una de las secciones de este capitulo, se describirá la implantación de los elementos mencionados.

3.1 FUNCION LOGARITMO.

El análisis de regresión es una de las técnicas mas usadas en la Estadistica, pues su propósito segun Kleinbaum es :

" Describir el espacio, dirección y forma de la relación entre variables independientes y una variable dependiente continua ".

Algunas veces, la relación buscada puede ser representada con un modelo no lineal, por ejemplo :

$$Y = EXP(a + bX)$$

donde :
a y b son constántes fijas y desconocidas,
X es la variable independiente y
Y es la variable dependiente.

El problema para el estadistico consiste en estimar a y b.

Un camino es :

ajustar el modelo
$Z = a + bX$
donde $Z = ln(x)$

Asi, transformando a la variable dependiente obtenemos un modelo lineal, facilitando la estimación de a y b.

El no incluir a la función logaritmo, obligaria al usuario del lenguaje a crear una rutina para evaluar a LN(X), rutina que puede ser ineficiente e inexacta comparada con la rutina proporcionada por PASCAL. También se consideró que la función logaritmo es la función inversa de la función exponencial y ésta es una función que ya existe en el lenguaje estadistico.

Además, como se explicará, fue un proceso simple la implantación de la función logaritmo.

ALGUNOS ELEMENTOS DE LA EVALUACION DE EXPRESIONES

Para completar la implantación de la evaluación de expresiones, se adicionaron los siguientes elementos :

--- Suma de cuerpos gráfica.
--- Inversa de matrices.
--- Determinante de matrices.
--- Función logaritmo.

En cada una de las secciones de este capitulo, se describirá la implantación de los elementos mencionados.

3.1 FUNCION LOGARITMO.

El análisis de regresión es una de las técnicas mas usadas en la Estadistica, pues su propósito segun Kleinbaum es :

" Describir el espacio, dirección y forma de la relación entre variables independientes y una variable dependiente continua ".

Algunas veces, la relación buscada puede ser representada con un modelo no lineal, por ejemplo :

$$Y = EXP(a + bX)$$

donde :
a y b son constántes fijas y desconocidas,
X es la variable independiente y
Y es la variable dependiente.

El problema para el estadístico consiste en estimar a y b.

Un camino es :

ajustar el modelo
$Z = a + bX$
donde $Z = \ln(x)$

Así, transformando a la variable dependiente obtenemos un modelo lineal, facilitando la estimación de a y b.

El no incluir a la función logaritmo, obligaria al usuario del lenguaje a crear una rutina para evaluar a LN(X), rutina que puede ser ineficiente e inexacta comparada con la rutina proporcionada por PASCAL. También se consideró que la función logaritmo es la función inversa de la función exponencial y ésta es una función que ya existe en el lenguaje estadístico.

Además, como se explicará, fue un proceso simple la implantación de la función logaritmo.

El primer paso fue crear el archivo PALCL8.EST añadiendo al archivo PALCL7.EST la palabra 'LN'. A continuación, se ejecutó el programa HPC2 para actualizar a los archivo PCVDT y PCCHAR.

Fue necesario modificar a la gramática, en primer lugar se incorporaron las siguientes producciones :

B(LN) = 2045
B(5022) = 5022

la primera para representar a la palabra 'LN' y la segunda corresponde a un llamado semántico. Otra producción modificada fue (PRIMARIO), agregándole la alternativa :

<LN> '(' (EXP) ')' <5022><5026>

En el analizador semántico se incluyó un llamado a la rutina MOPDOR (llamado 5022) encargado de meter al operador 'LN' en el stack de operadores; en el arreglo OPER (arreglo que contiene los nombres de las operaciones del lenguaje) se guardo el elemento 'LOG' y se colocó el llamado a la rutina OPUNARIA (llamado 5026) para generar el código consistente en un llamado a la rutina OPERUNA.

Las modificaciones a la rutina de biblioteca OPERUNA fueron minimas.

El operador logaritmo tiene las siguientes características :

PRECEDENCIA	OPERADOR	CODIGO Y NOMBRE
8	LN(exp)	22.- Logaritmo natural de la expresion real.

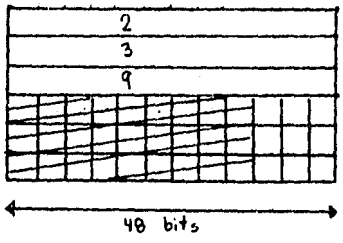
3.2 IMPLANTACION DEL TIPO CUERPO GRAFICA

En esta sección, se discutirá la representación y operación de las variables tipo cuerpo gráfica.

La representación del cuerpo de una gráfica consta de tres localidades destinadas para guardar las dimensiones de la gráfica, un número N de localidades en las que vivirá propiamente la gráfica y cuatro localidades para los máximos y mínimos de las abscisas y ordenadas de los puntos graficados.

En la búsqueda del ahorro en espacio, se decidió que cada punto de la gráfica ocupara cuatro bits en lugar de los 8 bits ocupados por un caracter.El número que se almacena en esos cuatro bits es un apuntador a la tabla TCAR, lugar donde vivirá realmente el caracter.

Como ejemplo representemos el cuerpo de una gráfica de tres por nueve :



Debe observarse que en la primer celda siempre se almacenará un dos y en las siguientes dos celdas vivirán las dimensiones de la gráfica.

Si el número de columnas de una gráfica es q, un renglón ocupará (q-1)div(12) + 1 celdas reales (en una celda real vivirán 12 puntos de una gráfica) y, si p es el número de renglones, el número de celdas ocupadas por los puntos de la gráfica son N = p*((q-1)div(12)+1) .

Este diseño aunque desperdicia espacio, es cómodo de trabajar y el ahorro con respecto a guardar un punto en ocho bits es considerable.

Para implantar el tipo cuerpo gráfica, fue necesario redefinir el tipo de los elementos que forman a la memoria, ahora TIPOMEM es :

```
TYPE
     TIPOMEM    = RECORD
           CASE TMULT OF
              RE : ( R : REAL);
             CAR : ( C : ARRAY[0..5] OF CHAR);
             GRA : ( G : ARRAY[0..5] OF BYTE);
          END;
```

La tabla de caracteres (TCAR)definida por :

```
     VAR  TCAR : ARRAY[ 0..15 ] OF CHAR ;
```

es manejada de la siguiente manera: se inicializa con TCAR[0]=' ', esto permite almacenar hasta 15 caracteres distintos para graficar. Cuando en una gráfica se usa un caracter c que no se ha ocupado, éste se guarda en TCAR (supongamos lugar I).

Entonces el caracter c será representado en la gráfica por el número I. El caracter ' ' (blanco) siempre vivirá en TCAR[0].

El lugar en el que se almacenarán los caracteres siguientes esta dado por la siguiente sucesión : 2, 3, 4,.. 15, 2, 3,..; de ésta manera se 'olvidan' los caracteres 'viejos'.

El cuerpo de una gráfica estaba definido en una alternativa de <PRIMARIO> como se puede observar :

```
<PRIMARIO>::= ... |'(' <EXP> <GRAF O RREAL ARR> ')' | ...
<GRAF O REAL ARR>::= ')'| ',' <EXP> ',' <EXP> ')'
```

ahora (PRIMARIO) tiene una alternativa para el cuerpo de una gráfica y otra para una expresión dentro de un paréntesis :

```
<PRIMARIO>::= ...| '<<'<5017><EXP><5034> ',' <EXP><5034>','
                   <EXP><5034> '>>'
                 | '(' <EXP> ')' |...
```

y se eliminó a < GRAF O REAL ARR>.

La rutina CGRAFICA (llamado 5017) tiene como tarea, generar código para crear un cuerpo gráfica temporal que se asignará o participará en una operación suma.

La primer tarea de CGRAFICA es la de generar código para que en ejecución se obtengan las dimensiones del cuerpo gráfica. Para esto revisa el descriptor del lado izquierdo de la asignación. Si existe un error en el uso de un cuerpo gráfica, se generará código para crear el cuerpo de una gráfica de dos por dos y otros elementos del compilador detectarán el error, por ejemplo la rutina ESCRIBE (explicada en 4.3) .

Despdes de generar el código para que en ejecución se determinen las dimensiones del cuerpo gráfica, se genera código para apartar e inicializar el espacio que ocuparán los puntos de la gráfica; el cuerpo de una gráfica es inicializado con blancos.

En seguida se genera el código que evaluará la primera expresión, y se verifica que el tipo del resultado sea Arreglo real (4). De no ser este el caso, se señala un error. De no existir error se genera la linea : GRAFX(LOC,NIV,TIPO) donde (LOC,NIV,TIPO) describen el resultado (en compilación) de la expresión. La rutina GRAFX debe obtener (en ejecución) la dirección donde viven las abscisas de los puntos a graficar, señalando un error y deteniendo la ejecución del programa si el número de dimensiones del arreglo resultante no es uno.

Las ordenadas de los puntos a graficar serán el contenido del arreglo unidimensional que resulte de evaluar a la segunda expresión, por lo que, recibe un tratamiento idéntico al de la primer expresión. La rutina GRAFY tiene un funcionamiento análogo a GRAFX.

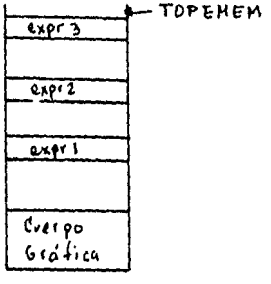
Por último, se genera código para evaluar a la tercera expresión y se mete al stack de operandos (en compilación) el operando correspondiente al cuerpo gráfica.

La tercera expresión proporciona el caracter con el cual se va a graficar, por lo que el tipo de la expresión (en compilación) debe ser Texto o Arreglo de textos (3 o 5), si no existe error en la evaluación de la tercera expresión es generado un llamdo a la rutina GRAFC(LOC,NIV,TIPO), donde (LOC,NIV,TIPO) describe al resultado (en compilación) de la tercera expresión.

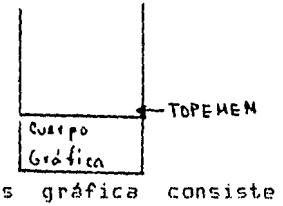
La rutina GRAFC se encarga de crear propiamente el cuerpo de la gráfica, calculando los máximos y mínimos de abscisas y ordenadas, y recuperar el espacio ocupado por los temporales.

En ejecución, el tipo de la tercer expresión debe ser Escalar texto (3) y pueden ser distintos los tamaños de los arreglos correspondientes a la primera y segunda expresión, graficándose el número de puntos para los cuales existen abscisas y ordenadas.

Supóngase que expr1,expr2 y expr3 son expresiones acordes con la definición del cuerpo de una gráfica, el estado del stack de memoria, antes de ejecutarse la rutina GRAFC es :



Cuando GRAFC construye el cuerpo gráfica, se recupera el espacio ocupado por los temporales y la memoria tendrá el siguiente aspecto :



La suma de cuerpos gráfica consiste escencialmente de encontrar los máximos y mínimos del cuerpo gráfica resultado, y para cada cuerpo gráfica que interviene en la suma se reconstruyen los puntos (usando los máximos y mínimos de la gráfica) y almacenarlos en el nuevo cuerpo gráfica, usando los nuevos máximos y mínimos.Sólo se permite la suma de cuerpos gráfica de iguales dimensiones.

Para almacenar y recuperar un punto de una gráfica se diseñaron dos rutinas : BGRAF y ALMGRAF.

La rutina BGRAF toma como parámetro un byte y un número entero que señalará cúales cuatro bits del byte serán decodificados, el encabezado de BGRAF es :

```
FUNCTION BGRAF( B : BYTE ; K : INTEGER) :INTEGER;
```

si K es par, BGRAF regresará el número (en decimal) formado con los cuatro bits más significativos, cuando K es impar se interpreta a los cuatro bits menos significativos.

La rutina ALMGRAF colocará en cuatro bits de un byte un número entre cero y quince, la declaración de ALMGRAF es :

```
PROCEDURE ALMGRAF ( VAR B:BYTE; K:INTEGER; VALOR :INTEGER);
```

donde B es el byte a ser modificado, K es interpretado de igual manera que por BGRAF y VALOR es el dato a almacenar.

La construcción de estas rutinas fue posible gracias a la facilidad de Turbo Pascal para manejar variables tipo byte, en otras versiones de Pascal puede simularse el manejo de bytes con caracteres.

3.3 INVERSA DE UNA MATRIZ

Uno de los procesos mas usados en la aplicación de la Estadística es el de obtener la inversa de una matriz dada, por lo que el lenguaje estadístico permite obtenerla directamente.

Siguiendo la definición original de García [G1], si A es una matriz de mxn, se define a la matriz inversa generalizada de A, llamémosle X a la matriz con las siguientes propiedades :

- AXA = A
- XAX = X
- AX es simétrica
- XA es simétrica

Puede probarse que X siempre existe y es única. Si A es cuadrada y no singular entonces $X = A^{-1}$ .Si A es rectangular y de rango completo entonces $X = (A'A)^{-1}A'$.

Para calcular X, se hace uso de la rutina SVD (Singular Value Decomposition) presentada por Forsythe [F1]. La rutina SVD descompone a la matriz A de mxn, como un producto de la forma

$$A = USV'$$

donde U es una matriz ortogonal de mxm, V es una matriz ortogonal de nxn y S es una matriz diagonal de mxn con $s_{ii} >= 0$ para i = 1 ..n, los números s_{ii} son llamados valores singulares y denotados por σ_i .

Sea S^+ definida por :

$$S(i,j) \begin{cases} 1/s_{ii} & \text{si } i=j \text{ y } s_{ii}>0 \text{ para } i=1..n \\ 0 & \text{en otro caso} \end{cases}$$

entonces puede probarse que X es igual a V S U'.

Por ser U y V invertibles (U'U = I), entonces rango(A) = rango(USV') = rango(S). Por lo tanto el rango de A es igual al número de elementos en la diagonal de S distintos de cero.

Es posible entonces que por errores de redondeo una matriz de rango incompleto tenga todos sus valores singulares distintos de cero. Es conveniente por lo tanto, trabajar con el rango efectivo definido como el número de valores singulares mayores que cierto nivel de tolerancia (T).

Si consideramos el rango efectivo de la matriz A, definimos a S_t^+ por :

$$S_t^+(i,j) \begin{cases} 1/s_{ii} & \text{si } i=j \text{ y } s_{ii}>T \text{ para } i=1..n \\ 0 & \text{en otro caso} \end{cases}$$

entonces, la matriz X definida por $X = V S_t^+ U'$ y llamada pseudo-inversa efectiva de la matriz A, tiene las siguientes propiedades :

```
-- XAX = X
-- AX es simétrica
-- XA es simétrica
-- ||  AXA - A  || <=  T
-- ||  X  || <= || W || para toda matriz W que
   cumpla las tres primeras condiciones.
```

La rutina de biblioteca para invertir matrices calculará la matriz inversa generalizada con un nivel de tolerancia igual a cero. Pudiendo el usuario modificarlo si inserta una linea al programa objeto, donde se asigne un valor a la variable TOOL, si el valor de TOOL es negativo, se trabajará con -TOOL.

Además, si se añade la linea 'PSEUDO:=TRUE;' se imprimirán los valores singulares de la matriz.

En el lenguaje estadístico la matriz inversa de una matriz X es denotada por X@.

En Forsythe [F1] se dan pautas para escoger el valor de TOOL como una función de la exactitud en los datos y la precisión de la computadora anfitriona.

Un problema común en Estadística es el de ajustar una recta por mínimos cuadrados, cuyo planteamiento es el siguiente :

Se desea encontrar $b \in R^p$, tal que minimice $\| Y - Xb \|$
Si $Y \in R^n$, X es una matriz de nxp, además de ser X y Y fijos y conocidos.
Puede probarse que la solución para b está dada por

$$\hat{b} = (X'X)^{-1} X'Y$$

Estimar b con $\hat{b}$ tiene dos inconvenientes : la matriz $X'X$ es muy mal condicionada generalmente y algunas veces no es invertible.

El uso de la rutina SVD en el cálculo de la matriz pseudo-inversa presenta los siguientes beneficios :

-- Eligiendo adecuadamente a TOOL es posible determinar la existencia real de $(X'X)^{-1}$.

-- El cociente $\sigma_{max}/\sigma_{min}$ es generalmente del mismo órden numérico que el número de condicion de la matriz X, por lo tanto desechando los valores singulares menores que TOOL, tiene el efecto de reducir el número de condición de la matriz X aproximadamente a $\sigma_{max}/\sigma_{min}$ aunque posiblemente aumente el tamaño de los residuales.

Si la matriz X es de rango completo, es posible calcular a la matriz $(X'X)^{-1}X'$ en el lenguaje estadístico de dos maneras :

1.- X@ 2.- ((X'><X)@)><X'

donde X' es la matriz transpuesta de X, recomendándose (1) porque en su cálculo sólo intervienen matrices ortogonales y diagonales. Por lo tanto en el lenguaje estadístico, b puede ser expresado como (X@)><Y .

3.4 DETERMINANTE DE UNA MATRIZ.

La rutina de biblioteca que calcula el determinante de una matriz verifica que la matriz sea cuadrada. De no suceder esto, se indica un error y se detendrá la ejecución del programa.Si X es una matriz cuadrada, la rutina DET transforma a X en una matriz diagonal superior y calcula su determinante como el producto de los elementos en la diagonal.

En el lenguaje estadístico, el determinante de una matriz X es expresado como ! X !.

En algunos problemas estadísticos importa el valor del determinante más que saber si es un número distinto o igual a cero.

Es posible modificar la cantidad con la cual se discriminan a los elementos distintos de cero, si se inserta en el programa objeto una línea en la que se asigne un valor a la variable TOOL.

3.5 MODIFICACIONES A LA IMPLANTACION DEL COMPILADOR

Referente a este capitulo se realizaron las siguientes modificaciones

-- Se incorporó la función logaritmo. Esto implicó modificar la gramática, los analizadores y las rutinas de biblioteca.

-- Los puntos de una gráfica son representados en cuatro bits en lugar de ocho bits.

-- La definición gramatical del cuerpo de una gráfica fue modificada para facilitar la compilación, diferenciando claramente una expresión dentro de un paréntesis, del cuerpo de una gráfica.

-- Se eliminaron los descriptores secundarios del descriptor de una gráfica donde se almacenabá a las localidades de los máximos y mínimos de las abscisas y ordenadas de los puntos graficados.

ELEMENTOS DE ENTRADA-SALIDA.

4.1 ARCHIVOS

En el lenguaje estadístico existen dos tipos de archivos, los asociados a un dispositivo de entrada-salida y los archivos en disco.

La declaración de una lista de archivos, queda sintácticamente definidada por las siguientes producciones :

```
<DECL>::= ... ! 'ARCHIVO' <4016> <TIPO ARCH> < L DE AR >!...
<L DE AR>::= <IDENTIF> '=' <TEXT><R DE ARCH>
<R DE ARCH>::= ','<IDENTIF> '=' <TEXT> ! e
<TIPO ARCH>::= 'LECTURA'!'ESCRITURA'! e
```

Todo archivo tendrá tres nombres, el interno para el compilador, el nombre en el programa para el usuario y el externo para el sistema operativo. El nombre en el programa corresponde a <IDENTIF>, el nombre externo será igual al texto definido por <TEXT> y el interno será determinado por el compilador.

El descriptor de un archivo consiste de un descriptor primario, cuyos campos son :

1) Apuntador a la tabla de cabezas de listas (LIB) y al diccionario de identificadores.
2) Archivo (10).
3) Archivo para lectura (1) o para escritura (2).
4) Nombre interno.
5) Apuntador al nombre externo.

La declaración de una lista de archivos es realizada por la rutina DECARCH (llamado 4016), encargada de crear un descriptor por cada archivo. Si la alternativa tomada en <TIPO ARCH> es 1 ('LECTURA'), la forma de los archivos será Archivo de lectura (1), en caso contrario la forma de los archivo será Archivo de escritura (2); la inclusión de <TIPO ARCH> en la gramática es una modificación realizada en el presente trabajo.

El nombre interno de un archivo se obtiene de la siguiente manera: si se trata de un archivo en disco, el nombre interno será igual al número de archivos en disco declarados anteriormente más cuatro. Así el primer archivo en disco que se declare tiene como nombre interno cuatro, si el archivo es un dispositivo de entrada-salida, tendrá un nombre predefinido.

Existen tres archivos especiales asociados con los dispositivos de entrada-salida, cuyas características se resumen en la siguiente tabla :

NOMBRE	NOMBRE INTERNO	DISPOSITIVO	FORMA
Impresora	Uno	Impresora	Escritura
Teclado	Dos	Teclado	Lectura
Video	Tres	Pantalla	Escritura

La importancia de los archivos en disco radica en que permiten conservar grandes cantidades de información.

Los archivos manejados en disco son archivos tipo texto y algunas de sus propiedades son :

--- Acceso secuencial.
--- Permiten almacenar números y caracteres.
--- Es posible modificarlos con un editor, facilitando la captura de datos.

En el momento de compilarse la primer proposicion de un nivel lexicográfico se revisan los descriptores del nivel actual y por cada archivo se genera la siguiente línea :

```
ASSIGN( XXX 'nombre-interno' , 'nombre-externo' );
```

Si la forma de un archivo es Lectura (1) se genera :

```
{$I-} RESET(XXX 'nombre-interno' );
```

Por el contrario, si la forma del archivo es Escritura (2), se genera :

```
{$I-} REWRITE(XXX 'nombre-interno' );
```

En cualquiera de los dos casos, en seguida se generará :

```
CHECA {$I+};
```

la rutina CHECA verificará que no exista error al abrir un archivo.De existir un error la rutina CHECA detendrá la ejecución del programa.

Después de ser declarado un archivo, éste vuelve a aparecer cuando es usado. El uso de un archivo es procesado por la rutina ARCHIVO, y ésta posee el siguiente encabezado :

```
PROCEDURE ARCHIVO ( LECTURA : BOOLEAN );
```

El valor de la variable LECTURA señalará cuando el archivo es usado en la proposición LEE o ESCRIBE.

La rutina ARCHIVO generará la línea :

NUMARCH:= 'nombre-interno';

NUMARCH es una variable entera del programa objeto, que indicará en ejecución al archivo que se desea accesar. ARCHIVO también deposita en la variable FILVAR a la literal formada por la concatenación de 'XXX' y el nombre interno; la variable FILVAR servirá para generar el código de las proposiciones LEE y ESCRIBE. Otra tarea realizada por ARCHIVO es verificar que la forma del archivo a usar esté acorde con el valor de la variable LECTURA.

La forma de un archivo debe ser única en todo un programa.

La implantación de archivos en el lenguaje estadístico usando PASCAL planteó dos alternativas :

--- Declarar un conjunto fijo de archivos por cada bloque, al igual que las etiquetas. Para preservar el diseño original del lenguaje, donde no se daba un orden para la declaración de los distintos objetos.

--- Declarar el número necesario de archivos por cada bloque, esta opción implica modificar a la semántica del lenguaje y añadiría la siguiente restricción :
" Los archivos serán los primeros objetos en ser declarados, en cualquier nivel lexicográfico ".

Se decidió usar la segunda alternativa, pues por cada archivo declarado se crea un buffer de aproximadamente 120 bytes, además de no limitar el número de archivos declarados en los programas escritos en el lenguaje estadístico.

4.2 FORMATOS

Un formato es una lista de caracteres de control, que señalarán la manera de interpretar los caracteres leidos o de editar los elementos a escribir.

La definición gramatical para la declaración de un formato está dada por :

```
<DECL> ::=  .. | 'FORMATO' <4017> < L DE FOR>| ....
<L DE FOR> ::= <DEF DE FOR><RF1>
<RF1> ::= ',' <DEF DE FOR><RF1>|e
<DEF DE FOR> ::= <IDENTIF> '=' <L DE TEXT>
<L DE TEXT> ::= <TEXT><RLC>
<RLC> ::= ',' <TEXT><RLC>|e
```

La declaración de un formato consiste del identificador asociado al formato y el formato en sí. El formato es considerado como la concatenación de los textos que forman a la lista de textos.

Un formato debe cumplir las siguientes reglas sintácticas :

```
<FORMATO> ::= <FOR><R FOR>
<R FOR> ::= ',' <FOR><R FOR>|e
<FOR> ::= <NUM><FOR S>|<FOR S>
<FOR S> ::= '(' <FORMATO> ')' |  'F'<NUM> '.'  <NUM>  |
            'I'<NUM> | 'E' <NUM> | 'C' <NUM> | 'X' | '/' |
            '$' <LETRAS> '$' | '%' <LETRAS> '%'
```

donde <LETRAS> es una cadena de caracteres, distintos de '$' si el delimitador es '$', o distintos de '%' si el delimitador es '%'.

La interpretación de un formato dependerá de si éste es usado en una proposición de lectura o escritura.

La rutina DECFORM (llamado 4017) creará para cada formato un descriptor con la siguiente información :

1) Apuntador a la tabla de cabezas de listas (LIG) y al diccionario de identificadores (VARDIC).
2) Formato (11).
3) Número de textos.
4) Localidad de CHARS donde vive el formato.
5) Cero.

El primer campo del descriptor primario es un apuntador al nombre del formato y en compilación no es revisada la sintáxis de los formatos; ésta se revisa hasta ejecución.

El uso de un formato es procesado por la rutina FORMATO. El encabezado de la rutina FORMATO es :

```
PROCEDURE FORMATO(
     VAR LOCF,NUMF:INTEGER;  VAR LIBRE,SIFOR:BOOLEAN  )
```

la rutina FORMATO regresará en LOCF y NUMF la descripción del formato (número de textos y localidad donde vive el formato) respectivamente. Cuando se uso el formato libre, la variable LIBRE tendrá el valor de Verdadero, y LOCF y NUMF serán iguales a cero, El único error (SIFOR es Falsa) detectado por FORMATO es aquél donde se usa como identificador de un formato a un identificador desconocido o que no corresponde a un formato.

La escritura de un número real será hecha por la rutina WRITE de PASCAL. La correspondencia entre los caracteres de edición y el correspondiente llamado a WRITE se presentan en la siguiente tabla :

FORMATO	PARAMETRO DE WRITE
Fw.d	:w:d
Ew	:w
Iw	:w:0

La descripción detallada y las restricciones de los formatos pueden leerse en el manual de Turbo Pascal [T1].

A continuación se mencionan algunas dea las reglas para el uso de formatos.

Una variable tipo texto debe ser escrita con el formato Cw, donde w es el ancho del campo, los caracteres son ajustados a la izquierda.

Un espacio en blanco es representado por 'X' y un salto de linea por '/'.

El formato Ew durante la lectura de un número real tratará de interpretar a los siguientes w caracteres como un número real.

Los números enteros pueden ser leidos con el formato Iw.

El formato Fw.d funciona igual que Ew si el número leído contiene '.' o 'E', si X es el número leído y no contiene '.' ni 'E' es interpretado como X*(10**(-d)) por Fw.d y X por Ew.

El formato Cw, servirá para leer una variable tipo texto con un campo de w caracteres.

Los formatos wX y w/ tienen la misma interpretación cuando se lee o escribe a terminal. Para archivos en disco los formatos wX y w/ son interpretados al igual que en FORTRAN.

Durante la ejecución del programa objeto, la rutina FORMATO revisará sintácticamente el formato, además de representarlo en un arreglo de enteros llamado FTO y definido por:

```
VAR FTO : ARRAY[ -134 .. -14 ] OF -134 .. 120;
```

cada formato tendrá una representación en el arreglo FTO, consistente en una serie de números, los formatos con su representación son mostrados en la siguiente tabla :

FORMATO	REPRESENTACION
rFw.d	-1 , r , w , d
rIw	-2 , r , w
rEw	-3 , r , w
rCw	-4 , r , w
rX	-5 , r
r/	-6 , r
r$(texto)$	-7 , r , l , d1 , d2
r@(texto)@	-7 , r , l , d1 , d2

En ella l es el número de caracteres que conforman al texto y d1 y d2 representan a partir de qué localidad en CHARS vive el texto.

Esta representación impone como restricciones :

--- Los factores de repetición (r) no pueden ser mayores a 120.

--- Un formato del tipo $(TEXTO)$ debe vivir en un solo texto y no debe contener a más de 120 caracteres.

De esta manera, los elementos de FTO son interpretados como se explica a continuación :

RANGO	INTERPRETACION
-134 .. -14	Apuntador a FTO
-13	Fin del formato
-12 .. -1	Formato
1 .. 120	Parámetro de un formato

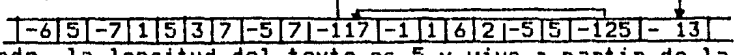
Como ejemplo representemos el siguiente formato :

5/ , $texto$, 7x , 3 (F6.2 , 5x)

simbólicamente es representado por :

/5 $1'texto' X7 3 (F 1 6 2 X5)

y numéricamente es representado por :



donde la longitud del texto es 5 y vive a partir de la localidad 3*120 + 7 de CHARS.

Para completar el manejo de formatos en ejecución se usan las variables CONTFTO e ICONTFTO definidas en :

```
VAR   ICONTFTO : INTEGER;
      CONTFTO  : ARRAY[ 1 .. MAXFTO ] OF INTEGER;
```

donde MAXFTO es una constante igual a cinco e ICONTFTO indica el nivel de anidación actual en el formato, CONTFTO[i] es el número de veces que falta por recorrer el subformato con un nivel de anidación i.

4.3 PROPOSICION ESCRIBE

Esta proposición permite emitir los resultados de un programa a un archivo o a un dispositivo de entrada salida.

La definición sintáctica de la proposición ESCRIBE es :

```
<P ESCRITURA> ::= 'ESCRIBE' 'EN' <5062> <IDENTIF> 'CON' <FORM>
                  <L DE ELEM A ESC)>
<L DE ELEM A ESC> ::= 'ESTO'<5033><EXP><5034><RE5>|e
<RE5> ::= ',' <5033><EXP><5034><RE5>|e
```

La rutina ESCRIBE (llamado 5061) es la encargada de generar el código correspondiente a la proposición ESCRIBE. Es necesario recordar que cuando se usa el formato libre y la expresión a escribir corresponde a un tipo estructurado, las rutinas de impresión se encargarán de distribuir adecuadamente los campos de la estructura.

La rutina ESCRIBE efectuará los siguientes llamados :

```
ARCHIVO(FALSE);
FORMATO(LOCF,NUMF,LIBRE,SIFOR);
```

explicados en 4.1 y 4.2 respectivamente. Después de ejecutarse las rutinas ARCHIVO y FORMATO, LOCF,NUMF,LIBRE y la variable FILVAR representan toda la información necesaria del formato y archivo usados.

Cuando la lista de elementos a escribir es vacía, se verifica que el formato usado no sea libre y se genera un llamado a la rutina :

```
ESCSEXP( LOCF,NUMF :INTEGER;VAR FILVAR :TEXT );
```

encargada de escribir un formato cuando no existen elementos a escribir.

Si la lista de elementos a escribir es no vacía, se genera código para evaluar cada elemento de la lista y, dependiendo del tipo del elemento y el formato usado, se genera el código adecuado, tal y como se presenta en las siguientes tablas :

FORMATO LIBRE

TIPO	RUTINA
1,2,3	ESCLIBRE(LOC,TIPO:INTEGER; VAR FILVAR :TEXT);
6	PINTAGRAF(LOC,TIPO:INTEGER; VAR FILVAR :TEXT);
7	PINTATAB(LOC,TIPO :INTEGER; VAR FILVAR :TEXT);
8	PINTAHIST(LOC,TIPO:INTEGER; VAR FILVAR :TEXT);

FORMATO NO LIBRE

TIPO	RUTINA
1,2,4	PINTAR(LOC,TIPO :INTEGER; VAR FILVAR :TEXT);
3,5	PINTAT(LOC,TIPO :INTEGER; VAR FILVAR :TEXT);

Cada elemento a escribir está caracterizado por (LOC,TIPO) y su nivel lexicográfico es pasado al código por medio de una asignación a la variable NIVELUNO.

Las combinaciones de TIPO y FORMATO que no aparecen en las tablas se consideran como errores.

En los siguientes párrafos se describirá brevemente el funcionamiento de cada una de las rutinas mencionadas.

La rutina ESCLIBRE escribe un escalar ya sea real o texto, con un llamado a la rutina WRITELN de PASCAL.

Las rutinas PINTAGRAF, PINTAHIST y PINTATAB obtienen la localidad en ejecución donde vive el objeto de interés y proceden a imprimirlo; los parámetros de estas rutinas con su significado se presentan en las siguientes tablas.

PARAMETRO	SIGNIFICADO
MAXCR	Número de caracteres en el registro de salida
LB	Largo de las barras en la impresión de Histográmas.
AB	Ancho de las barras en un Histograma.
SB	Separación entre barras de un Histograma.
NEX, NEY	Número de segmentos en los que se dividirán las escalas de abcisas y ordenadas en la impresión de gráficas.
AE	- Número de caracteres ocupados para imprimir a TITREN en la impresión de Tablas. - Número de espacios mínimos como margen izquierdo en la impresión de Histogramas. - Magnitud del campo con el que serán escritos los elementos de la escala en Y.

PARAMETRO	RANGO	VALOR INICIAL
LB	5 - 50	10
AB	2 - 20	8
SB	0 - 20	0
NEX	2 - 10	2
NEY	2 - 10	2
MAXCR	60 - 120	80
AE	5 - 20	10

El usuario podrá modificar el valor inicial de estos parámetros si inserta una línea en el programa objeto que les asigne un valor dentro de los rangos permitidos.

Fue modificada la semántica de MARCOREN y MARCOCOL; en la definición original se escribían las líneas correspondientes a MARCOREN y MARCOCOL si en la posición correspondiente existía un número distinto de cero. Ahora el criterio de decisión es la existencia de un número mayor que cero. Además, el número de líneas que ocupará un renglón de la tabla será igual a la parte entera del valor absoluto del elemento de MARCOREN y el formato con el que se escribirán los elementos de una columna será formado por los 2 primeros dígitos a la izquierda y a la derecha del punto decimal.

Así, si el elemento número tres de MARCOCOL es 5.02, los elementos de la tercer columna serán escritos con el formato F5.2. Si el elemeto de MARCOCOL es 5.2 los elementos de la tercer columna son escritos con F5.20. Debe tenerse el cuidado de asignar valores a MARCOREN y MARCOCOL, de lo contrario interpretará cualquier valor que encuentre, pudiendo ocasionar resultados imprevistos.

Cuando no se usó el formato libre, la rutina :

```
DATOSIO(LOC:REAL; NIV,TIPO:INTEGER;VAR LS,TS,NS:integer);
```

toma el operando en compilación (LOC,NIV,TIPO) y regresa la localidad en ejecución (LS), tipo (TS) y el número de escalares que forman al elemento a escribir. Si el tipo en compilación es Literal real (1), entonces LS será igual a TOPEMEM.

Algunas variables globales usadas son : NUMIO, igual al número de escalares que faltan por leer o escribir; NUMFOR, el número de escalares que serán escritos con el formato actual y RESTOFOR que es el número de escalares que faltan por escribirse con el formato actual.

La actualización de estas tres variables es hecha por la rutina DATOSFOR.

Además, LCS es una variable global que almacena la dirección en memoria del próximo escalar a escribirse.

La rutina PINTAR llama a la rutina DATOSIO para obtener el número de escalares reales a escribir (almacenándolo en NUMIO) y la localidad del primer escalar a escribir (guardándolo en LCS).

Inicializadas LCS y NUMIO se recorre el formato, obteniendo los formatos a usar y en función de éstos se llama a la rutina WRITE de PASCAL con los parámetros adecuados. Se dejará de recorrer el formato hasta que NUMIO es cero. PINTAR detendrá la ejecución del programa cuando se intente escribir un real con el formato para textos (formato Cw).

PINTAT tiene un funcionamiento completamente análogo a PINTAR.

4.4 PROPOSICION DE LECTURA

La proposición LEE sirve para transferir datos de un archivo a una variable declarada en un programa escrito con el lenguaje estadístico.

La definición sintáctica de la proposicion LEE es :

```
<P LECTURA> ::= 'LEE''DE'<IDENTIF><FIN ARCH>'CON'<FORM>
                <L DE RECEP >
<FINARCH>  ::= 'FIN'<IDENTIF>|e
<FORM>     ::= 'LIBRE'|<L DE TEX>|<IDENTIF>
<L DE RECEP> ::= 'ESTO'<EST REF><RL3>|e
<RL3 >     ::= ',' <EST REF><RL3>|e
```

El origen de los datos es definido por el archivo (identificador que sigue a 'DE'). La manera en que se interpretan los carcateres es proporcionada por el formato (ya sea libre, lista de textos o un formato representado con un identificador) y los objetos que servirán como destino de los datos leídos son aquéllos que forman la lista de elementos a leer.

Opcionalmente, cuando se usa la extensión fin de archivo, en la variable representada por el identificador que sigue a la palabra 'FIN' se almacenará un 1 si se ha encontrado el fin de archivo.

La variable mencionada debe ser un escalar real, sin ser parámetro por referencia.

La rutina LEE, al igual que la rutina ESCRIBE, llama a las rutinas ARCHIVO y FORMATO.

Los llamados hechos por LEE son :

```
ARCHIVO(TRUE);
FORMATO(LOCF,NUMF,LIBRE,SIFOR);
```

además, se invoca a la rutina FINARCH declarada como sigue :

```
PROCEDURE FINARCHIVO( VAR  FINARCH :BOOLEAN;
                      VAR  LOCF,NIVFA : INTEGER);
```

La variable FINARCH indicará con Verdadero si fue usada la extensión fin de archivo. De ser así, LOCFA, NIVFA representará a la variable escalar donde se alamacenará el número 1 si fue encontrado el fin de archivo.

El uso de una proposición LEE con una lista de receptores vacía, sirve para verificar si se ha encontrado el fin de un archivo. Cuando la lista de receptores es vacía y no se uso la extensión fin de archivo se señala un error.

La rutina :

```
LEEFA( LOCFA,NIVFA :INTEGER ; VAR FILVAR :TEXT);
```

tiene como tarea colocar en la variable representada por (LOCFA,NIVFA) el valor de 1 si se ha encontrado el fin del archivo FILVAR.

Cuando la lista de elementos a leer es no vacía, para cada uno de ellos se obtiene su posible tipo en ejecución y en función de éste, se generan los llamados presentados en las tablas.

El formato libre sólo sirve para leer variables escalares y el código generado dependerá de la aparición de la extensión fin de archivo.

FORMATO LIBRE

TIPO	FIN ARCHIVO	LLAMADO
2,3	NO	LLEESFA(LOC,NIV,TIPO:INTEGER;VAR FILVAR:TEXT);
2,3	SI	LLEECFA(LOCFA,NIVFA :INTEGER; LOC,NIV,TIPO:INTEGER;VAR FILVAR:TEXT);

El código generado también dependerá del uso de la extensión fin de archivo cuando el formato usado no es el libre; es posible leer escalares o arreglos ya sean de tipo real o texto.

TIPO	FIN ARCHIVO	LLAMADO
2,3 4,5	NO	LEESFA(LOC,NIV,TIPO:INTEGER;VAR FILVAR:TEXT);
2,3 4,5	SI	LEECFA(LOCFA,NIVFA:INTEGER; LOC,NIV,TIPO:INTEGER;VAR FILVAR:TEXT);

Nuevamente, por cualquier combinación de formato y tipo que no aparezca en algunas de las tablas se señala un error, (LOC,NIV,TIPO) representa al receptor en compilacion, (LOCFA,NIVFA) a la variable usada en la extensión fin de archivo y FILVAR al archivo de donde se leerán los datos.

La rutina básica de lectura cuando es hecha con formato, es la rutina LEECADENA, cuyo encabezado es el siguiente :

```
PROCEDURE LEECADENA(ES_NUMERO,ES_REAL:BOOLEAN;W:INTEGER;
                    VAR FILVAR :TEXT);
```

LEECADENA leerá los siguientes w caracteres del archivo FILVAR y los interpretará como un número si ES_NUM es igual a Verdadero, o como una cadena de caracteres si ES_NUM recibe el valor de Falso, cuando la variable ES_REAL recibe el valor de Verdadero los caracteres se interpretarán como un número real, de no ser así, deberán interpretarse como un número entero.

LEECADENA asignará el valor de Verdadero a la variable FINARCH, cuando se encuentre el fin de archivo.

Cuando los caracteres leídos no corresponden con la interpretación pedida, los caracteres se rechazan si son leídos del teclado o se detendrá la ejecución del programa si la lectura se realiza de un archivo en disco.

Si los caracteres leídos se interpretan como un número, el valor numérico de la cadena de caracteres es depositado en la variable BUFREAL; en caso contrario, son almacenados en la variable tipo cadena BUFSTRING. BUFREAL y BUFSTRING son variables globales declaradas en el programa objeto. Además si LEECADENA encuentra el fin de archivo, asignará el valor de Verdadero a la variable FINARCH.

Las rutinas LEEREAL y LEETEXT tienen como tarea leer variables tipo real y texto respectivamente con formato no libre.

La rutina LEEREAL, declarada como sigue :

```
PROCEDURE LEEREAL(LOCLEER,NUMLEER:INTEGER;VAR :FILVAR:TEXT);
```

asigna a LCL (Localidad a Leer) la primer localidad del objeto ser leído (pasada en LOCLEER) y a NUMIO el número de escalares que componen al objeto (pasado en NUMLEER), para después recorrer el formato con la ayuda de la rutina DATOSFOR, e ir leyendo uno a uno los escalares que forman al objeto por leer, LEEREAL detendrá la ejecución del programa cuando se intente leer una variable tipo real con un formato inadecuado o se encuentre el fin de archivo (FINARCH tiene el valor de Verdadero).

La rutina LEETEXT, tiene un funcionamiento análogo a LEEREAL; sirve para leer variables tipo texto y la declaración de la rutina es :

```
PROCEDURE LEETEXT(LOCLEER,NUMLEER:INTEGER; VAR FILVAR :TEXT);
```

La rutina LEESFA es la rutina central de lectura y su funcionamiento es explicado en los siguientes párrafos.

Si el receptor no es afectado por una expresión selectiva de arreglo, se invoca a la rutina DATOSIO para obtener el tipo, localidad, y número de escalares que forman al receptor. Si el tipo es Escalar real o Arreglo real (2 o 4) se llama a la rutina LEEREAL, si el tipo es tres o cinco Escalar texto o Arreglo de textos (3 o 5) se llama a la rutina LEETEXT.

Cuando el receptor es afectado por una expresión selectiva de arreglo, en primer lugar se efectúa el llamado : DECOMSELEC(FALSE) ; de esta manera es 'creado' en el tope de memoria un arreglo con el número de dimensiones y magnitudes necesarias y suficientes para almacenar a todos los elementos seleccionados del arreglo por leerse. Conocidas las dimensiones y magnitudes para el arreglo creado, se obtiene la primer localidad del arreglo creado y el número de escalares que lo forman.

El tipo del receptor determinará si se llama a la rutina LEEREAL (tipos escalar real o arreglo de reales) o a la rutina LEETEXT(tipos escalar texto o arreglo de textos) . Después de leer los datos y almacenarlos en el tope de la memoria, el llamado : RUTSEL(TRUE), asignará el arreglo 'creado' al verdadero arreglo receptor.

La rutina LEECFA invoca a la rutina LEESFA y después revisa que la variable FINARCH tenga el valor de Verdadero, para asignar el valor de 1 a la variable representada por (LOCFA,NIVFA).

La lectura con formato libre es traducida a :

```
READLN( FILVAR,BUFSTRING);
```

Si el receptor es un escalar real, el valor numérico de los caracteres leídos es almacenado en BUFREAL, para después asignar BUFREAL o BUFSTRING al objeto receptor. Estos son acciones realizadas por la rutina LLEESFA.

La rutina LLEECFA consiste de un llamado LLEESFA y la actualización de la variable representada por (LOCFA,NIVFA), si la variable FINARCH posee el valor de Verdadero.

4.5 MODIFICACIONES AL LENGUAJE

Las modificaciones al lenguaje implantadas en este capitulo son:

--- La interpretación de los campos MARCOCOL y MARCOREN cambió radicalmente. Aunque la interpretación ya no es tan simple, la interpretación actual permite definir el formato con el que se editarán los datos y el número de lineas que ocupará cada renglón del cuerpo de una tabla, características no consideradas en la definición original del lenguaje.

--- El formato libre puede ser usado para leer o escribir escalares; Esta es una ampilación del uso del formato libre.

--- El formato wX cuando es usado en una proposición de lectura del teclado, se interpreta como escribir w espacios en blanco a pantalla.

--- En el diseño original del lenguaje se pensaba dejar el manejo de los formatos al leguaje FORTRAN; en la implantación actual del lenguaje escrito en PASCAL, fue necesario emular a los formatos de FORTRAN. Buscando la simplicidad sólo se implantarón algunos formatos.

--- Se añadió la restricción semántica de que los archivos deben ser los primeros objetos en ser declarados.

--- Se modificó la definición sintáctica para la declaración de una lista de archivos.

RUTINAS

En el lenguaje estadístico existen dos tipos de rutinas, las rutinas sin tipo y las rutinas de tipo real. Las rutinas son importantes porque pueden servir para escribir programas compactos, sencillos y entendibles.

Una rutina en el lenguaje estadístico es traducida a un procedimiento en PASCAL. Esto añadió la siguiente restricción al lenguaje : 'las rutinas deben ser los últimos objetos en ser declarados en cualquier nivel lexicográfico', al igual que en PASCAL.

En las secciones de este capítulo se explicará la manera en que se genera el código para la declaración y uso de las rutinas.

5.1 DECLARACION

La declaración de una rutina consta de un encabezado, un cuerpo y la palabra 'FIN', que señala el fin de la rutina, en el encabezado se especifica si la rutina tendrá tipo y los posibles parámetros de ésta.

La definición sintáctica del encabezado de una rutina, está dada por :

```
<DECL>  ::= ....  |'RUTINA'<4018><Z TIPO><IDENTIF>
                    <ZONA DE PAR FOR>'INICIO'|...
<Z TIPO> :: 'REAL'|e
<ZONA DE PAR FOR> ::= '('<L DE PAR FOR>')'|e
<L DE PAR FOR>::=<ZONA TIPO VAL><L DE ID><RD11>
<RD11>::= ',' <ZONA TIPO VAL><L DE ID><RD11>|e
<ZONA TIPO VAL>::= 'VAL' 'REAL' | <TIPO>
<TIPO>::= <ARREG>| 'REAL' | 'TEXTO' | <ESTRUCTU>
<ARREG>::= 'ARREGLO'<TIPO ARR> '[' <NO DIM> ']'
<NO DIM>::=  ',' <NO DIM>|e
<TIPO ARR>::= 'REAL'<REN O COL>| 'TEXTO' | <REN O COL>
<REN O COL> ::= 'REN' | 'COL'
<ESTRUCTU>::= 'GRAFICA'|'TABLA'|'HISTOGRAMA'
```

La declaración de una rutina se realiza por la rutina DECRUTINA (llamado semántico 4018).

El nombre interno de una rutina será un número entero mayor que 0. Este número es obtenido de la varible NRUTINA, que es inicializada al principio del análisis semántico con 0 y tendrá como valor el número total de rutinas declaradas en el programa fuente. Por ello, la primera linea a generar es:

```
PROCEDURE RUTINA 'NOMBRE_INTERNO' ;
```

Y se crea el descriptor de la rutina tal y como se presenta en 1.4.1.

Después de declarar a la rutina, para cada parámetro de la rutina se crea un descriptor con la siguiente inf rmación:

1) Apuntador al diccionario de identificadores.
2) Tipo.
3) Forma.
4) Localidad en memoria, donde vivirá la dirección de la estructura de datos pasada como parámetro de referencia. Si es un escalar real pasado por valor la localidad donde vive.
5) cero o número de dimensiones, si se trata de un arreglo

La variable ILOCRE, indicará el desplazamiento relativo con respecto a la localidad donde comienzan a alamacenarse las variables locales a la rutina, ILOCRE es la localidad (a almacenarse en los descriptores) de la próxima variable a ser declarada.

Antes de invocar a ININIVEL se mete al stack de proposiciones -ILOCRE y Rutina (18).

A continuación se llama a la rutina ININIVEL encargada de reinicializar a los arreglos BNORCH, PROPOP, BARCHIVO, a las variables enteras ILOCRE,INDGT y NIVEL y a la tabla de símbolos, para procesar otro nivel lexicográfico. En este momento se declara nuevamente a la rutina para evitar que dentro de la rutina se declare otro objeto con el mismo identificador de la rutina que se está procesando. En seguida se declararán los parámetros de la rutina como si fueran variables declaradas en la rutina.

Los arreglos mencionados están definidos de la siguiente manera:

```
VAR     PROPOP : ARRAY [1..MAXNIVEL] OF BOOLEAN;
        BNOARCHIV : ARRAY [1.. MAXNIVEL] OF BOOLEAN;
        BARCHIVO : ARRAY [1..MAXNIVEL] OF RECORD
                                    PRIMERO,ULTIMO:INTEGER;
                          END;
```

La variable NARCH se inicializa al comenzar el análisis semántico con 4, e indicará el nombre interno del próximo archivo a ser declarado. Para recordar cuáles son los archivos declarados en el último bloque de nivel lexicográfico I, se usa el i-ésimo elemento del arreglo BARCHIVO. El campo PRIMERO del i-ésimo elememnto del arreglo BARCHIVO es inicializado con NARCH, pues NARCH es el posible primer archivo declarado en la rutina.

INDGT es inicializada con el valor 0 por ININIVEL. Así todas las etiquetas serán locales al bloque donde son declaradas.

La variable NIVEL es incrementada en 1 cada vez que inicia la compilacion de un nivel lexicográfico e indica como se mencionó, el nivel lexicográfico actual.

PROPOP es un arreglo que sirve para verificar que todas las declaraciones antecedan a las proposiciones; así PROPOP[i] es Verdadero si en el i-ésimo nivel lexicográfico, dentro del bloque actual ha sido declarada cuando menos una proposición.

BNOARCHIV tiene funciones similares a PROP, pues BNOARCHIV[i] Verdadero es si en el último nivel del bloque procesado del nivel i, dentro del bloque actual, ha sido declarado objeto que no es archivo.

BNOARCHIV y PROPOP son actualizadas y usadas por el procedimiento SEMANT, que es la rutina 'principal', pues ésta dirige el análisis semántico de cada árbol de derivación.

El arreglo BRUTINA también es actualizado desde SEMANT y es interpretado como sigue : BRUTINA[i] es el nombre de la última rutina con nivel lexicográfico i; por lo tanto, BRUTINA[1] es igual a 0, es decir el programa generado tiene como nombre interno 0.

Cada rutina junto con el programa principal deben tener una rutina asociada donde se inicializan las variables e inicializa el elemento del arreglo DISPLAY, correspondiente al nivel lexicográfico actual; estas rutinas se llaman INI 'NOMBRE_RUTINA', donde NOMBRE_RUTINA es el nombre de la rutina para la cual inicializará las variables.

Cuando aparece el primer objeto que no es archivo, se declaran los posibles archivos, se asigna al campo ULTIMO del elemento adecuado de BARCHIVO el valor NARCH-1, que es el último archivo declarado en el bloque, y se declaran también las variables enteras AUXTOP1T, AUXTOP2T, TOPEMEMT y DISPNIV. En seguida se genera:

```
PROCEDURE INI 'NOMBRE_RUTINA';
BEGIN
```

para declarar el procedimiento que inicializará las variables locales a la rutina.

Las variables AUTXOP1T, AUXTOP2T, TOPEMEMT y DISPNIV sirven para almacenar las cantidades necesarias para reestablecer el stack de memoria cuando termine de ejecutarse la rutina, las lineas generadas son:

```
AUXTOP2T := AUXTOP2;
AUXTOP1T := AUXTOP1;
TOPEMEMT:= TOPEMEM + 1;
DISPNIV := DISPLAY[ 'NIVEL' ];
```

Para inicializar al apuntador a la localidad donde comenzarán a almacenarse las variables locales al bloque, se escribe al código:

```
DISPLAY [NIVEL] := TOPEMEM ;
```

No es necesario guardar a la dirección de regreso, pues de este detalle se encargará el compilador de PASCAL.

Cuando el compilador detecta la primer proposicion de cualquier nivel lexicográfico son generadas las siguientes lineas:

```
BEGIN;
INI 'BRUTINAL NIVEL ]' ;
TOPEMEM := DISPLAY [ 'NIVEL' ] + 'ILOCRE';
```

En la segunda linea se llama a la rutina que inicializará las variables y en la tercer linea se actualiza el tope de memoria. También genera el código para que en ejecución se abran los archivos declarados en el programa escrito con el lenguaje estadistico.

Cuando el nivel lexicográfico es 1 (programa principal) justo antes del llamado a INIO es generado un llamado a la rutina INICIALI, encargada de inicializar con 0 todos los elemntos de la memoria, hacer TOPEMEM igual a 0, leer los archivos SEMVVDT y SEMCHARS e inicializar todas las variables que apoyan la ejecución de los programas generados.

Para compilar el cuerpo de una rutina se implantó el siguiente cambio: en compilación todas las variables son representadas por una tercia (LOC,NIV,TIPO). Ahora se da una nueva interpretación a NIV; si NIV>0 (LOC,NIV,TIPO) representa a una variable local al bloque o un parámetro por valor que vivirá en la localidad DISPLAY [NIV] + LOC; cuando NIV<0, (LOC,NIV,TIPO) representa a un parámetro por referencia, de manera que en DISPLAY[-NIV] + LOC, vive la dirección del objeto pasado como parámetro por referencia.

Con este diseño, las rutinas modificadas fueron : la rutina RESOPNDO que forma parte de las rutinas de biblioteca. RESOPNDO tiene como función obtener la localidad efectiva del objeto representado por (LOC,NIV,TIPO). En el analizador semántico fue necesario modificar a la rutina METEOPNDO, encargada de tomar un identificador que puede ser afectado por un campo y/o una expresión selectiva y obtener su representación en compilación, en una tercia (LOC,NIV,TIPO).

El fin de una rutina es compilado por el procedimiento FINRUTINA, FINRUTINA es llamada por CCOMP (cierra proposiciones compuestas) cuando el tope del stack de proposiciones es Rutina (18).

Las tareas realizadas por FINRUTINA son: sacar del stack de proposiciones el tope, recuperar el valor de ILOCRE, llamar a la rutina FINNIVEL y generar :

```
TOPEMEM := TOPEMEMT;
DISPLAY [NIVEL] := DISPNIV ;
AUXTOP1 := AUXTOP1T;
AUXTOP2 := AUXTOP2T;
```

que son las acciones necesarias para que en ejecución se reestablezca en el stack de memoria, el estado que tenía con anterioridad al llamado de la rutina.

El procedimiento FINNIVEL debe realizar las siguientes funciones:

--- Eliminar los descriptores de las variables locales al bloque y de los posibles parámetros formales de la rutina y reestablecer las listas que viven en LIG y LI.

--- Generar código para cerrar los archivos declarados en la rutina.

--- Revisar las tablas de etiquetas y señalar un error por cada etiqueta que tenga un valor igual o menor a 0.

--- Verificar que el stack de proposiciones sea vacio o el tope del stack sea Rutina (18) si se finaliza la compilación de una rutina o función.

Otro procedimiento que invoca a FINNIVEL es el procedimiento BFINSEMANTICA, que además de llamar a FINNIVEL, genera código para cerrar los archivos asociados con los dispositivos de entrada-salida, señala el total de errores en la compilación y cierra el archivo del monitoreo y el archivo donde se graba el código generado.

5.2 LLAMADO

En primer lugar se explica la manera en que se compila el llamado de una rutina para luego explicar las diferencias con la compilación de una función (rutina con tipo).

El llamado de una rutina es procesado por la rutina LLAMARUTINA (llamado semántico 5070). La definición sintáctica de un llamado a rutina está dado por:

```
<LLAMADO>::=<IDENTIF> <5070> <ZONA PAR>
<ZONA PAR>::=<ZONA DE PAR REALES>|e
<ZONA DE PAR REALES>::= '(' <LISTA DE PAR REALES> ')'
<LISTA DE PAR REALES>::=<EXP><5034><R14>
<R14> ::= ',' <EXP><5034><R14>|e
```

En primer lugar se verifica que el identificador corresponda a una rutina y que si existe la zona de parámetros reales la rutina tenga cuando menos un parámetro. De no ser así, se señala un error; si no existe la zona de parámetros reales y la rutina si tiene parámetros, se señala otro error. La rutina LLAMARUTINA genera :

```
AUXTOP2:= TOPEMEM;
```

```
AUXTOP2:= TOPEMEM;
```

para que en ejecución AUXTOP2 indique a partir de qué localidad comenzarán a almacenarse los parámetros de la rutina..

Cuando existen como menos un parámetro formal y un parámetro real, se llama a la rutina:

```
PASAPARAMETROS (FUNCION:BOOLEN;APDIC,NUMPAR:INTEGER);
```

donde FUNCION indicará si los parámetros corresponden a una función, APDIC es un apuntador a la torre de descriptores (DES), señalando dónde inicia la descripción de la rutina o función y NUMPAR es el número de parámetros formales de la rutina.

Finalmente, LLAMARUTINA genera las líneas:

```
TOPEMEM := TOPEMEM- 'Numero de parámetros' - 1;
RUTINA ' nombre de rutina' ;
```

La primer línea prepara a TOPEMEM para el llamado a la rutina y en la segunda se efectuá el llamado.

El procedimiento PASAPARAMETROS recorre la lista de parámetros reales hasta que ésta termina o procesa un número de parámetros igual a NUMPAR, genera código para evaluar cada expresión y, dependiendo del tipo resultante, se válida la correspondiencia entre los parámetros reales y formales, de acuerdo a la siguiente tabla. Es conveniente recordar que un operando temporal tendrá en su descripción el valor del campo LOC menor que 0.

PARAMETRO REAL		PARAMETRO FORMAL
TIPO	LOC	
1		Real por valor
2	< 0	Real por valor
2	> 0	Real por valor o referencia
3	> 0	Texto por referencia
4	> 0	Arreglo real por referencia
5	> 0	Arreglo de textos por referencia.
6	> 0	Gráfica por referencia.
7	> 0	Tabla por referencia.
8	> 0	Histograma por referencia.

Cualquier combinación que no aparezca en la tabla se considera como error. Además cuando el parámetro real es un arreglo, se verifica que coincida en el número de dimensiones con el parámetro formal y si el arreglo formal es un arrreglo columna o renglón, el parámetro formal debe tener la misma forma.

Cuando el parámetro es pasado por valor se genera un llamado a la rutina PASPV (Pasa por Valor) y si se trata de un parámetro pasado por referencia el llamado es a la rutina PASPR(pasa por referencia).

Los encabezados de PASPR y PASPV son:

```
PASPV(LOC :REAL;NIV,TIPO:INTEGER);
PASPR(LOC,NIV,TIPO:INTEGER);
```

PASPV debe colocar en el tope de memoria el valor del real pasado por valor y PASPR colocará en el tope del stack de memoria la direción del objeto pasado como parámetro por referencia.

También incrementan en 1 a AUXTOP2 e iguala a TOPEMEM y AUXTOP2.

Supóngase que (locx,nivx,tipox) representa a la variable real x en compilación y (locw,nivw,tipow) a la variable real w y se compila el siguiente programa :

```
INICIO;
     REAL X,W
     RUTINA R1 (REAL A,B)
     INICIO
     FIN;
     R1(X,W)
```

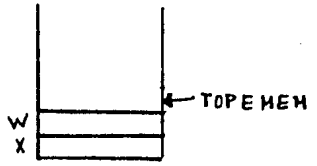
El código generado para la declaración y uso de la rutina es:

```
PROGRAM LENGEST;
{ programa en el lenguaje estadistico }
   procedure rutina1;  { se declara la rutina  }
   Var    topememt,auxtop1t,dispniv   :integer;
      { variables auxiliares de la rutina }

      procedure ini1;
      {  declara  a  las variables declaradas  en  el  lenguaje
         estadistico  }
      begin
         topememt := topemem  +  1; { actualiza topemem  y }
         auxtop1t := auxtop1;
         auxtop2t := auxtop2;
         dispniv := display[2];
         display [2] := topemem;
      end;
   begin  { de rutina1 }
      ini1;  { declara las variables }
      topemem := display[2] + 3;

      { Código para el cuerpo de la rutina }
```

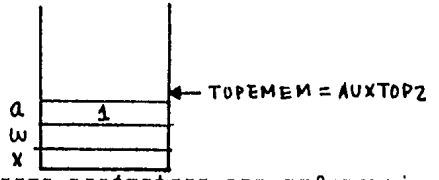
```
   display[2]:= dispniv; { reestablece el stack }
   auxtop1 := auxtop1t;
   auxtop2 := auxtop2t;
   topemem := topememt;
end;      { de rutina uno }


begin    { de programa principal }
   iniciali;
   ini0;
   topemem := display[1] + 3;       (1)

   auxtop2:= topemem;
   PASPR( locx,nivx,tipox);         (2)
   PASPR( locw,nivw,tipow);
   topemem := topemem - 3;          (3)
   rutina1;                  { se llama a rutina1 }
```

Los estados por los que pasa el stack de memoria son:

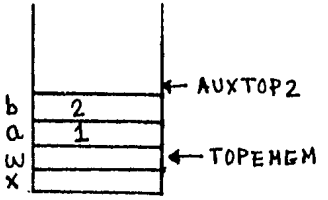
1.-Se han declarado a las varibles X y W apartando 2 localidades.



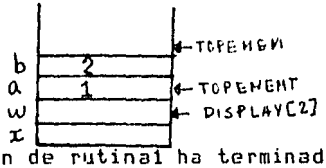
2.-PASPR obtiene la localidad donde vive X y la coloca en el tope de la memoria.



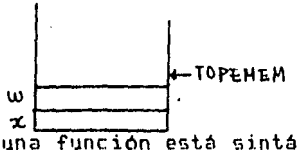
3.-Ya se ha pasado como parámetros por referencia a X y W; se prepara a topemem para el llamado a la rutina.



4.-Se invoca a rutina1 y el stack de memoria esta listo para ejecutar las proposiciones de la rutina1.



5.-La ejecución de rutina1 ha terminado y se restablece a topemem, auxtop1 y display[2].



El llamado de una función está sintácticamente definido por:

<PRIMARIO>:= ...| <IDENTIF><5032><ACPV>|...
<ACPV> :=".* <OP EST>| <ZONA DE PAR REALES>|e

Cuando el identificador corresponde a una función, se mete al stack de operandos (en compilación) un operando que represente a la localidad temporal donde se almacenará el resultado de la función. La localidad temporal donde se almacenará el resultado de la función es el tope de memoria; entonces la línea :

AUXTOP2:= TOPEMEM + 1;

deja una localidad para colocar el valor resultante, para después pasar a los parámetros.

Si el identificador no corresponde a una función y la alternativa en <ACPV> es igual a 2 se señala un error.

A partir de este momento el trato que recibe una función es igual al de las rutinas.

Después de haber generado el código para pasar los parámetros e invocar a la rutina, se escribe al código :

METE (TOPEMEM-1,2);

para que en ejecución se meta al stack de operandos la localidad efectiva donde vivirá el resultado de evalúar a la función.

5.3 MODIFICACIONES AL LENGUAJE .

Con el fin de no complicar en mucho la implantación del lenguaje, se aceptaron las siguientes restricciones, que no se consideran excesivas.

--Si en una proposición de escritura con formato se invoca a una función, ésta no debe contener una proposición de lectura o escritura con formato.

--Si en la construcción del cuerpo de una gráfica se llama una función, dentro de ésta no se debe construir otro cuerpo gráfica.

El violar éstas restricciones ocasionará que se detenga la ejecución del programa. No son detectados en compilación.

Las modificaciones al diseño original del lenguaje son:

-Se añadió la restricción de que las rutinas deben ser los últimos objetos en ser declarados.

-Las rutinas pueden ser recursivas, característica no considerada en el diseño original.

MODIFICACIONES PROPUESTAS
AL COMPILADOR Y AL LENGUAJE.

El desarrollo del compilador hasta este momento comprende todas las características del lenguaje estadístico.

Por razones de tiempo y en la búsqueda de una implantación no compleja, durante el desarrollo del compilador se dejó de lado en algunos momentos aspectos como la eficiencia o consistencia.

El compilador ya es un programa cuyo manejo puede aprender fácilmente cualquier persona que conozca un lenguaje de programación. Sin embargo, puede resultar un poco tedioso el manejo del compilador para una persona acostumbrada a los paquetes estadísticos.

Señalo a continuación algunos aspectos con los cuales considero mejore el compilador.

i) Las rutinas de biblioteca deben compilarse para cada programa en PASCAL generado por el compilador. Para resolver este problema existe una opción inmediata : cambiar la implantación a una versión de PASCAL para PC, que permita compilar las rutinas de biblioteca sólo una vez (por ejemplo PASCAL U.C.S.D. o Turbo Pascal v4).

ii) Turbo-Pascal permite mover bloques de bytes, proceso mucho más rápido que una asgnación de PASCAL. El uso de esta característica aumentaría la eficiencia de las rutinas de biblioteca.

iii) No se inicializan las variables de cada nivel lexicográfico, pero puede implantarse el siguiente procedimiento: en compilación es conocido el número de celdas reales ocupadas por las variables locales al bloque (supongamos V) y el número de parámetros de una rutina (supongamos P); entonces puede generarse código para que inicialice en cero V localidades a partir de la localidad TOPEMEM + P. Esta inicialización debe hacerse momentos después de actualizar el tope del stack , en la entrada a cada nivel lexicográfico.

iv) Pueden eliminarse las siguientes restricciones :

--- No poder construir el cuerpo de una gráfica en una rutina con tipo, cuando esta es invocada en la construcción del cuerpo de otra gráfica.

--- No poder usar una proposición de entrada-salida en una rutina con tipo, cuando esta es invocada desde una proposición de entrada-salida.

v) Debe permitirse que se aniden cualesquiera número de subformatos en un formato, para resolver este problema, los contadores que viven en el arreglo CONTFTO, deben vivir en el arreglo FTO.

vi) El reconocedor sintáctico no discrimina entre los distintos tipos de errorea. Es deseable modificar al analizador sintáctico o construir otro, para que el analizador sintáctico haga un mejor diagnostico de los errores .

vii) Podría crearse un archivo con los mensajes de error. Cuando se encuentre un error se lee del archivo el mensaje y se emite. La presencia del archivo puede ser opcional.

viii) Para buscar un átomo en el diccionario de símbolos arreglos CHARS y VARDIC), el método de busqueda es lineal, que se vuelve ineficiente cundo el diccionario es grande. Se podría implantar un método de Hash que vuelva más eficiente la búsqueda.

ix) Puede eliminarse el diccionario de textos en el programa objeto, si cada vez que se necesite a una constante de textos se genera una asignación del texto a la variable BUFSTRING. Por ejemplo si se necesita al texto 'HOLA', debe ser generado :

```
BUFSTRING := 'HOLA';
```

y en seguida, debe generarse código que almacene a BUFSTRING a partir de cualquier localidadad de memoria.

x) Los tamaños de las estructuras de datos usadas en el analizador semántico deberían ser escogidos de manera óptima.

xi) En el análisis semántico los números viven en el arreglo CHARS, representados como una cadena de caracteres. Para mejorar el manejo de los números puede crearse un diccionario de reales donde vivirán los números.

xii) Durante la evaluación de una expresión, el espacio ocupado por los resultados temporales no es reutilizado. Si el problema de capacidad de la memoria llega a ser grande, puede implantarse el siguiente procedimiento :

Definir a el arreglo ESPLIBRE como sigue :

```
VAR ESPLIBRE : ARRAY[1.. MAXLIBRE] OF
                              RECORD
                                LOCL,LONGL : INTEGER
                              END;
```

De esta manera, cada vez que saquemos un elemento del stack de operandos (en ejecución), podemos conocer la localidad y el número de celdas ocupadas por éste, y almacenar esta información en algun elemento del arrreglo ESPLIBRE.

Durante la ejecución, si conocemos el (los) operando(s) y el operador es posible conocer la longitud en celdas reales del operando resultante, se revisarían todos los elementos ocupados del arreglo ESPLIBRE, y si el operando resultante cabe en el espacio dejado por algún temporal, ahí debe ser almacenado; de no ser así, el operando resultado debe almacenarse a partir del tope de la memoria.

xiii) Cuando se imprime una Tabla, Gráfica o Histograma, existen parámetros para las rutinas de impresión, sería deseable que a estos parámetros se les diera un valor desde el lenguaje.

Por ejemplo, la proposición :

ESCRIBE EN ARCHIVO1 CON LIBRE(10,20,30) ESTO TABLA1;

escribiría en el archivo ARCHIVO1 a la Tabla TABLA1 y se asignaría la tercia (10,20,30) a los parámetros de la rutina encargada de imprimir a las estructuras tipo Tabla.También pueden definirse como variables pre-declaradas a los parámatros de las rutinas de impresión.

xiv) Las rutinas que imprimen Tablas, Gráficas o Histogramas son rudimentarias en el sentido de que no hacen uso de la graficación en alta resolución, podrían reescribirse las rutinas para mejorar la presentación del compilador.

xv) Cuando se opera con escalares el código generado es ineficiente, puede escribirse un optimizador de código.

xvi) Puede añadirse una opción de compilación para asignar valores a las variables TOOL y PSEUDO, usadas en el cálculo del determinante e inversa de una matriz, o bien TOOL y PSEUDO pueden definirse como variables pre-declaradas.

xvii) Dentro del código que se genera para declarar a una Tabla, éste podría inicializar a los campos MARCOREN y MARCOCOL.

xviii) Se puede optimizar el almacenamiento del cuerpo de una gráfica.

xix) Puede modificarse al analizador semantico para que en compilación se revise la sintáxis de los formatos, esta modificación consistiria en copiar la rutina que revisa al formato al analizador semántico.

CONCLUSIONES.

El lenguaje estadístico ha sido implantado totalmente y el compilador ya es un programa que puede usar fácilmente cualquier persona que conozca un lenguaje de programación.

El traslado del compilador a las microcomputadoras PC fue un factor decisivo para la conclusión de este trabajo, pues el compilador había alcanzado tal magnitud, que una microcomputadora como la FRANKLIN-ACE dificultaba el manejo e implantación del compilador.

Las microcomputadoras PC ofrecieron mayor memoria (tanto primaria como secundaria), mayor velocidad en procesamiento y compilación, un manejo limpio de archivos y un mejor editor.

A cambio, se perdió un aspecto que consideró importante en el uso del lenguaje, es el que las rutinas de biblioteca no pueden ser ligadas, ocasionando que tengan que compilarse por cada programa generado por el compilador.

Se han mencionado varios aspectos para lograr una versión mejorada del compilador, pero en mi opinión los aspectos más importantes a desarrollar son :

--- Lograr que las rutinas de biblioteca puedan ser ligadas a los programas generados por el compilador.

--- Hacer más explícitos los mensajes del analizador sintáctico.

También se puede avanzar en la búsqueda de eficiencia del compilador y del código generado, pues en algunos momentos se escogió el algoritmo a usar por su facilidad en la implantación.

En la parte del compilador desarrollada por este trabajo, se trató de programar estructuradamente y comentar adecuadamente los programas.

Por último, si el presente trabajo ayuda a un estudiante a entender mejor el proceso de compilación ,o las técnicas estadísticas, el presente trabajo habrá cumplido con demasía su objetivo.

APENDICE A.
OPCIONES DE COMPILACION.

La sintáxis de una opción de compilación es :

< Opción de Compilación > ::= '$$' < Letra > < Señal >
< Letra > ::= 'A' | 'C' | 'D' | 'E'
< Señal > ::= '+' | '-'

Una opción de compilación no debe aparecer dentro de un comentario. La linea donde aparece una opción de compilación es ignorada a partir de la aparición de ésta y no debe estar contenida en medio de una frase sintáctica.

La señal ('+' o '-') indicará cuando se activa o desactiva la opción de compilación; en un principio todas las opciones estan desactivadas.

El efecto de activar a las diferentes opciones de compilacion se explica a continuación :

$$A Impresión del árbol sintáctico

Por cada frase se imprime el árbol sintáctico. Este contiene para cada nodo la siguiente información : nombre, número, alternativa y un apuntador al nodo hermano a la derecha y de no existir éste, un apuntador al nodo ascendiente derecho más próximo.

$$C Impresión del texto limpio

Para cada frase se imprime su representación con un conjunto de números. Esta representación es el denominado texto limpio.

$$D Monitoreo del análisis semántico

Ocasiona la emisión de un reporte (bastante detallado) de todas las actividades realizadas por el analizador semántico.

$$E Monitoreo de la ejecución del programa

Origina la emisión de un reporte donde se describe a los operandos que intervienen en las operaciones, además de mostrar el contenido de algunas estructuras auxiliares.
Para su activación es necesario que ésta aparezca después de la primer proposición de cualquier nivel lexicográfico.

Todas estas opciones de compilación fueron herramientas para la construcción del compilador.

APENDICE B.
ERRORES EN COMPILACION Y EJECUCION.

Los errores detectados durante la compilación por el analizador semántico son :

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 desbordo.

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 esta mal declarada. El límite inferior es mayor al superior.

9. El valor del número o constánte 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 una constante.

14. Demasiadas dimensiones en la declaración de un arreglo . Máximo = 10.

16. Un arreglo que tenga más de una dimensión no puede declararse como arreglo renglón.

18. Expresión demasiado compleja. El stack de operandos se derramó.

19. ERROR DEL COMPILADOR.
El stack de operandos se encuentra vacío.
Probablemente en la expresión a analizar existe un identificador no declarado o de un tipo inadecuado.

20. Expresión demasiado extensa. El stack de operadores está lleno.

21. ERROR DEL COMPILADOR.
El stack de operadores se encuentra vacío.

22. Se trata de sumar al cuerpo de una gráfica una estructura de de distinto tipo.

23. Combinación inválida de operandos en una operación binaria : una estructura real con una estructura texto.

24. Los operadores unarios (más unario, negación, inversa, determinante y transpuesta) sólo se pueden aplicar a estructuras de tipo real.

25. Operación binaria inválida. Entre estructuras de tipo texto o cuerpo gráfica solo se puede aplicar la suma (+).

26. Operación unaria inválida. A las estructuras de tipo texto no se les puede aplicar operadores unarios.

27. En operaciones binarias sólo pueden participar operandos que sean escalares, arreglos, campos de estructura con forma de escalar o arreglo y cuerpos gráfica.

28. Variable inválida en las expresiones de un 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 asignación no puede aparecer una constánte o el identificador de una rutina.

32. Declaración de arreglos.El tamaño de cada dimensión debe ser cuando menos de dos.

33. No es permitido que un arreglo explícito aparezca dentro de otro arreglo explícito.

34. Demasiadas expresiones para construir a un arreglo explícito.Eliminar expresiones o simplificarlas.

35. No se permite combinar expresiones de tipo real con expresiones de tipo texto en la definición de un arreglo explícito

36. Demasiadas expresiones en la expresión selectiva de un arreglo. Eliminar expresiones o simplificarlas.

37. En la expresión selectiva de arreglo apliacada a una variable a la que se le va a asignar una expresion, 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 una estructura que no es arreglo.

39. Dentro un arreglo explícito la repetición de paréntesis debe ser positiva.

40. Expresión selectiva de arreglo. Se trata de tener acceso a un elemento con índice 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 de 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 una gráfica a otra variable, que es de distinto tipo o viceversa.

49. Etiqueta que fue utilizada pero que nunca apareció en el bloque. Sólo se pueden hacer transferencias incondicionales estrictamente locales al bloque.

50. Error fatal. Demasiadas etiquetas en el programa fuente. Tabla para manejar etiquetas saturada.

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.

56. Final inesperado del programa o bloque. Quedan proposiciones pendientes.

57. Terminación inesperada del programa fuente.

60. Un arreglo renglón pasado como parámetro por referencia debe tener una dimension. Se declara con una dimensión.

61. Error de tipo en el paso de parámetros.

63. Faltan parámetros en el llamado a una rutina.

64. Sobran parámetros en el llamado a una rutina.

65. Identificador que no corresponde a rutina sin tipo.

66. Referencia a identificador que no es rutina con tipo.

68. Operando cuyo tipo no le permite participar en uuna expresión.

69. Una rutina sin tipo no puede ser destino de una expresión.

70. Arreglo real como resultado en una expresión de la proposición REPITE CON o selectiva (LA ..) la expresión debe resultar en un escalar real.

71. El resultado de las expresiones REPITE CON o selectiva debe ser un escalar real.

72. La variable de control en la proposición REPITE CON debe ser una variable escalar real local al bloque o parámetro por valor.

73. El resultado de una expresión usada como condición debe ser de tipo real.

74. Etiqueta ya declarada.

75. Etiqueta usada que no es posible declarar en este nivel de anidación de proposiciones.

76. Etiqueta que ya no es posible declarar. Declararla implicaría un brinco al cuerpo de una proposición compuesta.

77. Anidación incorrecta de la proposición REPITE.

78. ERROR DEL COMPILADOR en el manejo del stack de proposiciones.

79. No se esperaba un 'SINO'.

80. Anidación incorrecta de proposiciones.

81. Identificador no corresponde a un formato.

82. Identificador no corresponde a un archivo.

83. Archivo de entrada , que intenta ser usado como archivo de salida.

84. Archivo de salida , que intenta ser usado como archivo de de escritura.

85. Variable usada en la extensión fin de archivo de la proposición LEE , debe ser escalar real sin ser parámetro por referencia.

86. Intento de escribir una lista de elementos vacía con formato libre.

87. Intento de escribir una variable no escalar, no Tabla, no Gráfica y no Histograma con formato libre.

88. Uso inadecuado del formato fijo. No sirve para escribir Gráfica, Tablas o Histogramas.

89. La lista de elementos a leer es vacía y no fue usada la extensión fin de archivo.

90. No es válido leer cuerpos de Gráficas.

91. Tipo de variable no permitido como receptor en la proposición LEE.

92. El formato libre sólo sirve para leer escalares.

93. Falta la palabra 'INICIO', con la que debe comenzar todo programa.

94. No se permite declaración alguna, después de aparecer la primer proposición.

95. Proposición 'INICIO' mal utilizada, sólo se emplea al inicio del programa.

96. Los archivos deben ser los primeros objetos en ser declarados.

97. Las rutinas deben ser los últimos objetos en ser declarados.

98. En este lugar del programa el identificador de una rutina con tipo (función) no debe aparecer del lado derecho en una proposición de asignación.

110. La expresión que proporciona a las coordenadas en X, para la construcción del cuerpo de un gráfica debe ser un arreglo real.

111. La expresión que proporciona a las coordenadas en Y, para la construcción del cuerpo de un gráfica debe ser un arreglo real.

112. La expresión que proporciona al caracter para la construcción del cuerpo de una gráfica debe ser de tipo texto.

Los siguientes son errores detectados por el analizador lexicográfico :

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 sintáxis.

103. Error fatal. La frase es demasiado grande. El buffer para almacenar al texto limpio se saturo.

104. Error de sintáxis en un número.

105. Error fatal. Demasiados identificadores, textos y números en el programa fuente. El arreglo CHARS del diccionario de átomos quedo 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.

Los siguiente errores son señalados durante la ejecución de los programas generados por el compilador.

Algunos de estos errores (indicados con *) detienen la ejecución del programa.

Para los errores que no detienen la ejecución del programa se indica la acción que realiza el código generado.

0. Caracteristica aun no implantada.

6. 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 número de dimensiones.

*8. Proposición de asignación. La expresión a asignar y el operando son de distinto tipo.

*9. Se trata de asignar a una estructura de tipo escalar, otra estructura que no es de tipo 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 transpuesta a un arreglo de más de dos dimensiones.

*13. Se trata de operar con dos arreglo 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 vacio.

*20. Es inválido tener acceso a un subarreglo de una estructura de tipo escalar en una expresión.

*21. El número de dimensiones de 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 el límite superior de la dimensión.

23. Indice a arreglo que es menor que 1. Se toma a uno como el índice.

24. Indice a arreglo, mayor al límite superior de la dimensión. Se toma dicho límite.

25. Pareja invalida de indices a arreglo . 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 de ser tipo real. Se toma como falsa a la expresión.

*30. La expresión que proporciona al caracter para la construcción de la gráfica debe ser un escalar tipo texto.

*31. El número de dimensiones de los arreglos que proporcionan a las abcisas y ordenadas para la construcción de la gráfica deben ser unidimensionales.

*32. Error fatal.Error en las rutinas de biblioteca.

*33. El parámetro pasado por valor debe ser un escalar real.

*34. Intento de sumar cuerpos de gráficas de distintas dimensiones.

*35. Intento de asignar un cuerpo de gráfica de dimensiones distintas al cuerpo gráfica receptor.

*36. No fue posible abrir un archivo. Se señala el error correspondiente al sistema operativo MS-DOS.

37. El largo de un texto dentro de un formato no debe ser mayor a 120 caracteres. Se toman 120 caracteres.

*38. Formato demasiado grande.Usar dos proposiciones LEE o ESCRIBE.

*39. Error en sintaxis de formato. Se escribe el caracter donde se detuvo el reconocimiento y su número dentro del formato.

*40. Formato con demasiados niveles de anidación (5). Simplificarlo.

*41. Lectura de un caracter no imprimible de un archivo.

*42. Cadena de caracteres leídos que no puede ser interpretada como un número.

*43. Formato para números reales, usado en la lectura de variables tipo texto.

*44. Formato para textos usado en la lectura de variables reales.

*46. Intento de escribir una variable tipo texto con un formato para números reales.

*47. Intento de escribir una variable real con un formato para escribir textos.

*48. La expresión que inicializará a la variable de control en la proposición REPITE CON debe ser escalar real.

*49. La expresión que proporciona el límite superior o el incremento en la proposición REPITE CON debe ser escalar real.

*50. La expresión que servirá para tomar la alternativa en la proposición selectiva debe ser escalar real.

*51. No es posible usar un formato en una rutina con tipo, cuando ésta es invocada en una proposición de entrada-salida.

*52. No es posible construir el cuerpo de una gráfica en una rutina con tipo, cuando ésta es invocada desde una expresión en la que se construya el cuerpo de una gráfica.

*53. El formato libre no sirve para escribir arreglos.

*54. No fue posible encontrar la inversa de una matriz; el algoritmo se detuvo en 30 iteraciones.

*55. El valor asignado a una función deber ser un escalar real.

97. Valor numérico que es mayor al máximo entero posible :
+ - 32767. Se toma el máximo valor posible.

99. División entre cero se toma como resultado el 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.

101. Logaritmo de un argumento no positivo .
Es decir : LN(a) con a<=0
Se toma como resultado a 0.

APENDICE C.
GRAMATICA.

Las convenciones para escribir a la gramática son :

i) Los símbolos no terminales deben ser encerradas por paréntesis.

ii) Para definir un símbolo no-terminal con una producción que contenga a símbolos terminales y no terminales se usa P( ).

iii) Para la definición de un símbolo no terminal en un llamado semántico, en un tipo de átomo o palabra reservada se usa B( ).

iv) Los símbolos terminales se encierran entre apóstrofes (un apóstrofe se representa con dos apóstrofes seguidos).

v) El fin de una producción es indicado con un punto y coma ';'.

vi) El signo de igual '=' se emplea como separador entre los dos lados de una producción.

vii) Las producciones que tengan igual antecedente deben ser escritas de la siguiente manera : se escribe el antecedente , el signo de igual y los consecuentes separados por el signo de admiración '!'; si una alternativa es prefijo de otra,el antecedente más grande debe ser el primero en escribirse.

viii) El fin de la gramática es expresado con un asterisco '*'.

ix) El símbolo de porcentaje '%', indica que existe un comentario en la gramática que termina con el final de la linea.

x) La producción vacía se representa sin símbolo alguno.

xi) Los rangos en las producciones tipo B, son interpretados como se explica a continuación.

Si B(X) = Y;

Rango de Y	Interpretacion de X
1010	Atomo numérico.
1020	Atomo tipo identificador.
1040	Atomo tipo texto.
2000-3999	Palabra Clave.
4000-9999	LLamado semántico.

En este trabajo las convenciones para representar a la gramática son :

.- Los símbolos no terminales son encerrados con '<' y '>'.

.- Los símbolos terminales se escriben entre comillas (').

.- El antecedente y consecuente de un producción son separados por '::='.

.- Las distintas alternativas de una producción son separadas con '|'.

.- La producción vacía es indicada con 'e'.

En las siguientes páginas se presenta el listado del archivo GTO.EST, que contiene la gramática usada por el compilador.

```
%                          Grámatica.

%      ********************* PALABRAS CLAVE Y ATOMOS ********************

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;
B(LN)=2045;
B(LECTURA)=2046;
B(ESCRITURA)=2047;
```

```
B(4010)=4010;
B(4011)=4011;
B(4012)=4012;
B(4013)=4013;
B(4014)=4014;
B(4015)=4015;
B(4016)=4016;
B(4017)=4017;
B(4018)=4018;
B(5001)=5001;
B(5007)=5007;
B(5010)=5010;
B(5014)=5014;
B(5015)=5015;
B(5017)=5017;
B(5018)=5018;
B(5019)=5019;
B(5020)=5020;
B(5021)=5021;
B(5022)=5022;
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;
B(5050)=5050;
B(5051)=5051;
B(5052)=5052;
B(5061)=5061;
B(5062)=5062;
B(5070)=5070;
```

```
%******* DECLARACIONES **********

P(PRINCIPIO)=(INIOVAC)(P) ';';
P(P)= (DECL) ! (5042) (L ETIQ) (P GRAL)(5047) !  ;
P(INIOVAC) = (INICIO) !  ;
P(P GRAL) = (COND) ! (ITER) ! (SELEC) ! (P INC);
P(RFI) = (HAS) ! (RC1);
P(L ETIQ) = (NUMERO) ':' (L ETIQ) ! ;
P(P INC) = (PR INC)(TERMINA);
P(TERMINA)=(FIN)(5045)(RFI)(TERMINA) ! ;
P(PR INC) = (ASIG) ! (INICIO) ! (LLAMADO) !
            (P VER) ! (P LECTURA) ! (P 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)(4016)(TIPO ARCH) (L DE AR) !
        (RUTINA)(4018)(Z TIPO)(IDENTIF)(ZONA DE PAR FOR)(INICIO)!
        (FORMATO) (4017) (L DE FOR);
P(TIPO ARCH)= (LECTURA)!(ESCRITURA)!   ;
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) !  ;

%              PROPOSICIONES
%
%*************** 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)(5050)(CUERPO ITER);
P(CUERPO ITER)=(INICIO) !
 (P INC)(HAS)(TERMINA) !
 (CON)(IDENTIF)':='(5033)(EXP)(5034)(SUMANDO)(5033)(EXP)(5034)(HASTA)
                    (5033)(EXP)(5034)(ESTO)(P INC)!
 (MIENTRAS)(5033)(EXP)(5034)(ESTO)(P INC) ;
P(HAS)=(HASTA)(5051)(5033)(EXP)(5034);

                %**************     VER     **********************

P(P VER)=(VER)(5043)(NUMERO);

                %****************   LLAMADO    ***************

P(LLAMADO)=(IDENTIF)(5070)(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(P LECTURA)=(LEE)(DE)(5061)(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(P ESCRITURA)=(ESCRIBE)(EN)(5062)(IDENTIF)(CON)(FORM)(L DE ELEM A ESC);
P(L DE ELEM A ESC)=(ESTO)(5033)(EXP)(5034)(RE5) !  ;
P(RE5)= ',' (5033)(EXP)(5034)(RE5) !  ;

                %*********** SELEC     ************************

P(SELEC)=(LA)(5052)(5033)(EXP)(5034)(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)='+' ! '-' ! (0) ;
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) ')' !
            '!' (EXP) '!' (5021)(5026) ! (ARREGLO EXP) !
          (LN) '(' (EXP) ')' (5022) (5026)                !
     '<<' (5017) (EXP) (5034) ',' (EXP) (5034) ',' (EXP) (5034) '>>' ;
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)(5034)(R14) ;
P(R14) = ',' (EXP)(5034)(R14) !  ;
*
```

APENDICE D
LISTA DE LLAMADOS SEMANTICO

Los llamados semánticos efectuados desde el programa principal son :

Llamado	Rutina	Función
4010	DECCTE	Declara una constánte real o texto
4011	DECESC	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 uan lista de Histogramas
4016	DECARCH	Declara una lista de archivos
4017	DECFORM	Declara una lista de formatos
4018	DECRUTINA	Declara a un rutina
5033	PROCEXP	Procesa una expresión
5035	ASIGNACION	Compila una proposicion de asignación
5042	DECET	Procesa la aparición de una etiqueta
5043	BRINCA	Compila una proposición de transferencia incondicional
5044	CONDICIONAL	Procesa la primer parte de una proposición condicional
5045	CCOMP	Cierra una proposición compuesta
5046	SECCON	Procesa a una alternativa de condicional o a la alternativa no etiquetadad de la proposición selectiva
5047	CSIMPLE	Cierra las proposiciones sencillas que deben ser cerradas por el punto y coma ';'
5050	REPITE	Compila el encabezado de la proposición iterativa
5051	FINREP	Procesa el fin de la proposición REPITE CON
5052	DECCASE	Compila el encabezado de la proposicion selectiva
5062	ESCRIBE	Compila a la proposición ESCRIBE
5061	LEE	Compila a la proposición LEE
5070	LLAMARUTINA	Procesa el llamado de una rutina sin tipo

Los llamados añadidos dentro de la rutina PROCEXP fueron dos, el llamado 5017 y el 5022.

Llamado	Rutina Ejecutada	Función
5017	CGRAFICA	Genera código para construir una gráfica y mete al stack de operadores el operando que represente al cuerpo de una grafica
5022	METEOPNDO	Mete el operando que resulta de aplicar la función logaritmo

APENDICE E.

En este apéndice se presentan cuatro programas donde se muestran las principales caracteristicas del lenguaje estadístico

Programa : PROG40.EST

```
INICIO

%       SE TIENE UN CONJUNTO DE DATOS QUE REPRESENTA PESOS MEDIDOS
  EN KILOGRAMOS DE 20  PAQUETES CONTENIENDO PARTES DE MAQUINARIA.

   1.- SE DESEA ELABORAR UNA TABLA DE FRECUENCIAS CON UN ANCHO DE
       CLASE DE DOS UNIDADES , CON LIMITE INFERIOR IGUAL A 92.

   2.- HISTOGRAMA PARA LAS FRECUENCIAS SIMPLES Y ACUNULADAS.

   3.- LOS POLIGONOS DE FRECUENCIAS SIMPLES Y ACUMULADAS.

-----------------------------------------------------------------;

ARCHIVO LECTURA
            PESOS    = 'PESOS.DAT';

ARCHIVO
            LISTADO = 'LP40.DAT';

CONSTANTE
    W               = 2,
    LIMITEINFERIOR = 92,
    NUMINT          = 8,
    MAXDATOS        = 40;

REAL
    NUMDATOS;

ARREGLO
    DATOS                       [ 1 : MAXDATOS],
    FRECSIMPLES,FRECACUM        [ 1 : NUMINT];

ARREGLO TEXTO
    LIMITES                     [ 1 : NUMINT];

ARREGLO
    LIMITR                      [ 1 : NUMINT];

TABLA
    TABFREC [ NUMINT , 2 ];

HISTOGRAMA
    HFREC[ NUMINT ];
```

```
GRAFICA
    GFREC[ 10, 20  ];

RUTINA LEEDATOS( ARREGLO REAL [ ] DATOS
                 REAL             NUMDATOS
INICIO
REAL
    I,FINARCHIVO;


FORMATO
   F = 'F5.2,/';
   FINARCHIVO:=0;
   I:=1;
   LEE DE PESOS FIN FINARCHIVO CON F ESTO DATOS[1];
   REPITE MIENTRAS NO FINARCHIVO ESTO
   INICIO
      I:=I+1;
      LEE DE PESOS FIN FINARCHIVO CON F ESTO DATOS[I];
   FIN;
   NUMDATOS:=I-1;
FIN;   %  DE LEEDATOS;

RUTINA ARMAFRECUENCIAS ( ARREGLO[] DATOS  REAL NUMDATOS
                         ARREGLO [] LIMITES,F,FF       )
INICIO
REAL
    I,LI;

    LI := LIMITEINFERIOR;
    % SE CALCULAN FRECUENCIAS SIMPLES        ;

    REPITE CON I:=1 SUMANDO 1 HASTA NUMINT ESTO
    INICIO
       LIMITES[I]:= LI;
       F[I]:= ( (LI<=DATOS[1_NUMDATOS])       Y
             (DATOS[1_NUMDATOS]<LI+W) )[ , ];
       LI:=LI+W;
    FIN;

    % SE CALCULAN FRECUENCIAS ACUMULADAS

    FF[1]:=F[1];
    REPITE CON I:=2 SUMANDO 1 HASTA NUMINT ESTO
       FF[I]:= FF[ I-1 ] + F[ I ];

FIN;   %   DE ARMAFRECUENCIAS

     % SE LEEN LOS DATOS;

     LEEDATOS(DATOS, NUMDATOS);

     % SE ARMAN LAS FRECUENCIAS;

     ARMAFRECUENCIAS(DATOS,NUMDATOS,LIMITR ,FRECSIMPLES,FRECACUM);
```

```
LIMITES := [[ NUMINT :' 92 - 94 ', ' 94 - 96 ', ' 96 - 98 ',
            ' 98 - 100', '100 - 102', '102 - 104','104 - 106'
            ,'106 - 108'  ]];
% SE ARMA LA TABLA DE FRECUENCIAS;

TABFREC.TITULO := 'TABLA DE FRECUENCIAS ';
TABFREC.NOMCOL := [[ 2 : 'FREC. SIMPLE','FREC. ACUMULADA']];
TABFREC.TITREN :='LIMITES';
TABFREC.TITCOL :='';
TABFREC.NOMREN := LIMITES;
TABFREC.CUERPO[ ,1 ] := FRECSIMPLES;
TABFREC.CUERPO[ ,2 ] := FRECACUM;
TABFREC.MARCOREN     := [[ NUMINT : NUMINT(-2) ]];
TABFREC.MARCOCOL     := [[ 2 : 2( 6.02)        ]];

% SE ESCRIBE LA TABLA;

ESCRIBE EN LISTADO CON LIBRE ESTO TABFREC;

% SE ARMA EL HISTOGRAMA DE FRECUENCIAS SIMPLES;

HFREC.TITULO := 'HISTOGRAMA DE FRECUENCIAS SIMPLES';
HFREC.CUERPO := FRECSIMPLES;
HFREC.NOMREN := LIMITES;

% SE ESCRIBE EL HISTOGRAMA DE FREC. SIMPLES ;

ESCRIBE EN LISTADO CON LIBRE ESTO HFREC;

% SE ARMA EL POLIGONO DE FRECUENCIAS SIMPLES ;

GFREC.TITULO := 'POLIGONO DE FRECUENCIAS SIMPLES ';
GFREC.CUERPO := << LIMITR,FRECSIMPLES,'*' >>;
GFREC.TITREN := 'FRECUENCIAS SIMPLES';
GFREC.TITCOL := 'X';

% SE ESCRIBE EL POLIGONO DE FRECUENCIAS SIMPLES ;

ESCRIBE EN LISTADO CON LIBRE ESTO GFREC;

% SE ARMA EL HISTOGRAMA DE FRECUENCIAS ACUMULADAS ;

HFREC.TITULO := 'HISTOGRAMA DE FRECUENCIAS ACUMULADAS';
HFREC.CUERPO := FRECACUM;
HFREC.NOMREN := LIMITES;

% SE ESCRIBE EL HISTOGRAMA DE FRECUENCIAS ACUMULADAS;

ESCRIBE EN LISTADO CON LIBRE ESTO HFREC;

% SE ARMA EL POLIGONO DE FRECUENCIAS ACUMULADAS;

GFREC.TITULO := 'POLIGONO DE FRECUENCIAS ACUMULADAS';
GFREC.CUERPO := << LIMITR ,FRECACUM,'*' >>;
GFREC.TITREN := 'FRECUENCIAS ACUMULADAS';
```

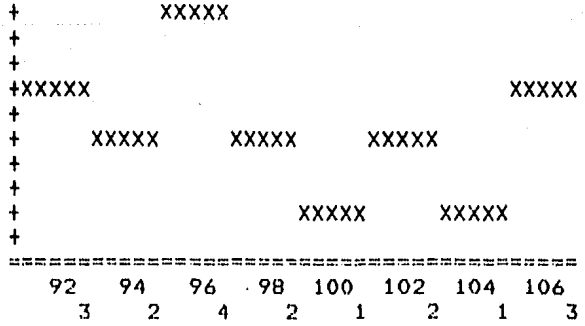
```
     GFREC.TITCOL := 'X';
     % SE ESCRIBE EL POLIGONO DE FRECUENCIAS ACUMULADAS;
     ESCRIBE EN LISTADO CON LIBRE ESTO GFREC;
FIN;  % DE PROGRAMA    ;
```

Archivo de Salida del Programa : PROG40.EST

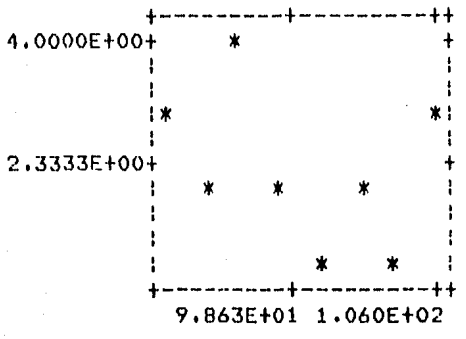
TABLA DE FRECUENCIAS

LIMITES	FREC. S	FREC. A
92 - 94	3.0	3.0
94 - 96	2.0	5.0
96 - 98	4.0	9.0
98 - 100	2.0	11.0
100 - 102	1.0	12.0
102 - 104	2.0	14.0
104 - 106	1.0	15.0
106 - 108	3.0	18.0

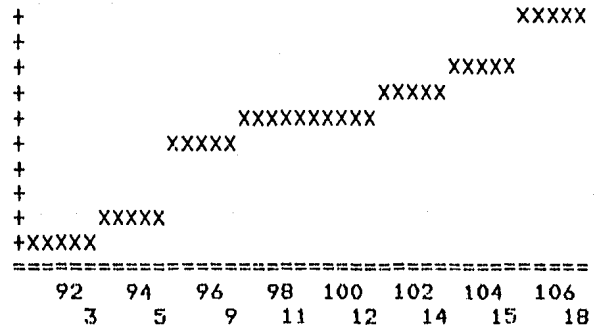
HISTOGRAMA DE FRECUENCIAS SIMPLES



POLIGONO DE FRECUENCIAS SIMPLES

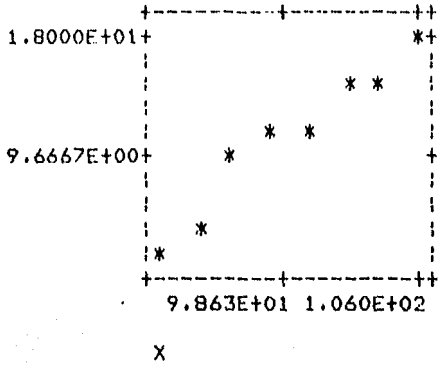


HISTOGRAMA DE FRECUENCIAS ACUMULADAS



POLIGONO DE FRECUENCIAS ACUMULADAS

FRECUENCIAS ACUMULADAS



```
     Programa : PROG41.EST


INICIO
%    LAS PUNTUACIONES DE UNA PAREJA DE JUGADORES SON RECOLECTADAS
  DURANTE 10 EVENTOS.
  SE  DESEA SABER SI EXISTE UNA RELACION ENTRE SUS PUNTUACIONES Y
  SE  DECIDE   PROBARSE LA HIPOTESIS DE INDEPENDENCIA  ENTRE  LAS
  PUNTUACIONES DE LOS JUGADORES.

    Ho   :   LAS PUNTUACIONES DE LOS DOS JUGADORES SON
           INDEPENDIENTES.

    Ha   :  EXISTE TENDENCIA A APAREAR PUNTUACIONES ALTAS DEL
          JUGADOR 1 CON PUNTUACIONES ALTAS DEL JUGADOR 2 O
          VICEVERSA.

SEA ALFA = 0.05 LOS VALORES DE W .025 Y W.975 SON -.6364 Y .6364
       RESPECTIVAMENTE.

-----------------------------------------------------------------;

ARCHIVO  LECTURA
                 PUNTOS  = 'PUNTOS.DAT';

ARCHIVO
                 LISTADO ='LP41.DAT';

CONSTANTE
     MAXDATOS = 20,
     ALFA = 0.05,
     W025 = -0.6364,
     W975 =  0.6364;

REAL
     NUMDATOS,
     VALORRHO;

ARREGLO
     PUNTOS1,RANGOSP1,RANGOSP2,PUNTOS2 [1 : MAXDATOS];

RUTINA LEEDATOS( ARREGLO []     PUNTOS1,PUNTOS2
                 REAL           NUMDATOS
INICIO
    REAL
        I,FINARCHIVO;
    FORMATO
        F = ' 3X,2I5,/';

    FINARCHIVO := 0;
    I := 1;
    LEE DE PUNTOS FIN FINARCHIVO CON F ESTO PUNTOS1[I],PUNTOS2[I];
```

```
    REPITE MIENTRAS NO FINARCHIVO ESTO
    INICIO
       I:= I+1;
       LEE DE PUNTOS FIN FINARCHIVO CON F ESTO
                                    PUNTOS1[I],PUNTOS2[I];
    FIN;

    NUMDATOS := I;
FIN; % DE LEEDATOS ;

RUTINA ASIGNARANGOS (ARREGLO [ ]    DATOS, RANGOS
                     REAL           NUMDATOS          )
INICIO
     REAL   I ;

     ESCRIBE EN LISTADO CON
                          * 10X,$==================$,/,',
                          * 10X,$No !  Dato  ! Rango$,/,' ,
                          * 10X,$------------------$,/' ;

     REPITE CON I :=1 SUMANDO 1 HASTA NUMDATOS ESTO INICIO
        RANGOS[I] := ( DATOS[I]>DATOS[1_NUMDATOS] ) [ . ]    +
                  ( (DATOS[I]=DATOS[1_NUMDATOS]) [ . ] + 1)/2 ;
        ESCRIBE EN LISTADO CON '10X,I3,F9.3,F7.2,/' ESTO
          I,DATOS[I],RANGOS[I];
     FIN;

     ESCRIBE EN LISTADO CON '10X, $==================$,4/';

FIN;  % DE ASIGNA RANGOS;

RUTINA REAL RHO ( ARREGLO [ ] RANGOX ,RANGOY
                  REAL        NUMDATOS              )
INICIO

    RHO:=1-6*((( RANGOX[1_NUMDATOS]-RANGOY[1_NUMDATOS])**2)[.])/
                 (NUMDATOS*(NUMDATOS**2 -1));
FIN;  % DE RHO;

     % SE LEEN LOS DATOS;
     LEEDATOS(PUNTOS1,PUNTOS2,NUMDATOS);

     % SE ASIGNAN RANGOS A PUNTOS1 Y PUNTOS2;
     ESCRIBE EN LISTADO CON
            '5/,10X,$Observaciones con sus$,/,10X,',
            '$Rangos asociados$,4/';
     ESCRIBE EN LISTADO CON '10X,$Jugador No 1 $,/';
     ASIGNARANGOS(PUNTOS1,RANGOSP1,NUMDATOS);
     ESCRIBE EN LISTADO CON '10X,$Jugador No 2 $,/';
     ASIGNARANGOS(PUNTOS2,RANGOSP2,NUMDATOS);

     % SE CALCULA  EL VALOR DE RHO;
```

```
    VALORRHO := RHO( RANGOSP1, RANGOSP2, NUMDATOS );

    % SE HACE LA PRUEBA ( DOS COLAS );

    ESCRIBE EN LISTADO CON
     '3(/),7X,$  EL VALOR DE LA ESTADISTICA RHO DE SPEARMAN ES : $,
       F6.2,2/,',
       ' 7X,$ POR LO TANTO CON UN NIVEL DE SIGNIFICANCIA DE $,
       F4.2,2/'
    ESTO VALORRHO,ALFA;

    ESCRIBE EN LISTADO CON
    '7X,$ Los cuantiles son :   $,2F6.3,/ ' ESTO W025,W975;

    SI (VALORRHO<W025) O ( VALORRHO>W975) ENTONCES
       ESCRIBE EN LISTADO CON '7X,$ Se rechaza Ho: $,/,11X,',
  '$Existe evidencia para suponer que las puntuaciones$,/,11X,',
       '$de los jugadores son dependientes. $,2/'
    SINO
     ESCRIBE EN LISTADO CON ' 20X,$ No se rechaza Ho:  $,/,25X,',
     '$Existe evidencia para suponer que las puntuaciones$,/,25X,',
      '$de los jugadores son independientes. $,2/'
FIN; % FIN DE PROGRAMA ;
```

Archivo de Salida del Programa : PROG41.EST

Observaciones con sus
Rangos asociados

Jugador No 1

No	Dato	Rango
1	147.000	4.00
2	158.000	7.50
3	131.000	2.00
4	142.000	3.00
5	183.000	9.00
6	151.000	5.00
7	196.000	10.00
8	129.000	1.00
9	155.000	6.00
10	158.000	7.50

Jugador No 2

No	Dato	Rango
1	122.000	4.00
2	128.000	9.00
3	125.000	8.00
4	123.000	5.50
5	115.000	2.00
6	120.000	3.00
7	108.000	1.00
8	143.000	10.00
9	124.000	7.00
10	123.000	5.50

EL VALOR DE LA ESTADISTICA RHO DE SPEARMAN ES : -0.60

POR LO TANTO CON UN NIVEL DE SIGNIFICANCIA DE 0.05

Los cuantiles son : -0.636 0.636
No se rechaza Ho:
Existe evidencia para suponer que las puntuaciones
de los jugadores son independientes.

```
     Programa : PROG42.EST

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 POR MINIMOS
       CUADRADOS.

   2.- CALCULAR EL COEFICIENTE DE DETERMINACION.

   3.- OBTENER LA TABLA DE ANDEVA ( ANALISIS DE VARIANZA ).

-----------------------------------------------------------------;

ARCHIVO  LECTURA
                 PUNTOS     = 'REGRE.DAT';

ARCHIVO
                 LISTADO    = 'LP42.DAT';

CONSTANTE
    MAXDATOS = 20;

REAL
    NUMDATOS,    ALFAEST,      BETAEST,       R2,     XMED,   YMED,
    SUMACUADREG, SUMACUADTOT, SUMACUADRESID;

ARREGLO
    X1,Y1 [ 1 : MAXDATOS ];

TABLA
    ANDEVA [ 3, 4];

RUTINA LEEDATOS ( ARREGLO [ ]  XX ,YY
                  REAL         NUMDATOS  )
INICIO

REAL
   I,FINARCHIVO;

FORMATO
   F = 'I3,E6,/';

    FINARCHIVO :=0;
    I:= 1;
    LEE DE PUNTOS FIN FINARCHIVO CON F ESTO XX[1],YY[1];
    REPITE MIENTRAS NO FINARCHIVO ESTO
    INICIO
       I:= I+1;
       LEE DE PUNTOS FIN FINARCHIVO CON F ESTO XX[I],YY[I];
    FIN;
```

```
    NUMDATOS := I;
FIN;  % DE RUTINA LEEDATOS ;

    % SE LEEN LOS DATOS;

    LEEDATOS(X1,Y1,NUMDATOS);

    % SE CALCULAN LOS ESTIMADORES POR MINIMOS CUADRADOS DE
      ALFA Y BETA ;

    XMED:= ( X1[1_NUMDATOS] ) [ . ]/NUMDATOS;
    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 ESCRIBEN LOS ESTIMADRES DE ALFA Y BETA;

    ESCRIBE EN LISTADO CON
        '3/,20X,$ Los parametros estimados son : $,2/,',
        ' 20X,$ Alfa  =  $,F8.3,/,20X,$ Beta = $,F8.3,/'
        ESTO  ALFAEST,BETAEST;

    % SE CALCULAN LAS SUMAS DE CUADRADOS;

    SUMACUADREG :=
    ( ( ALFAEST + BETAEST*X1[1_NUMDATOS] - YMED)**(2)) [ . ];
    SUMACUADTOT := ((Y1[1_NUMDATOS] - YMED)**2)[.];
    SUMACUADRESID := SUMACUADTOT - SUMACUADREG;

    % SE CALCULA Y ESCRIBE EL COEFICIENTE DE DETERMINACION;

    R2 := SUMACUADREG/SUMACUADTOT;
    ESCRIBE EN LISTADO CON
   '20X,$El coeficiente de determinacion es :$,F8.5,2/' ESTO R2;

    % SE ARMA LA TABLA DE ANDEVA;

    ANDEVA.TITULO  := '  TABLA DE ANDEVA ( ANOVA ) ';
    ANDEVA.NOMCOL  := [[ 4 : 'Gr. de libertad','S. de cuadrados',
                      'S. de cuad. medios',' F '              ]];
    ANDEVA.TITREN := 'Fte. de variacion ';
    ANDEVA.NOMREN  := [[ 3 : 'Regresion','Residual','Total'    ]];
    ANDEVA.CUERPO[ ,1]  := [[ 3 : 1,(NUMDATOS-2),(NUMDATOS-1) ]];
    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;
    ANDEVA.MARCOCOL := [[ 4 : 4(16,02) ]];
    ANDEVA.MARCOREN := [[ 3 : 3(-4,0) ]];
```

```
    % SE ESCRIBE LA TABLA;
    ESCRIBE EN LISTADO CON LIBRE ESTO ANDEVA;
FIN:  % FIN DE PROGRAMA;
```

Archivo de Salida del Programa : PROG42.EST

Los parametros estimados son :

Alfa = 4.138
Beta = 0.100
El coeficiente de determinacion es : 0.92698

TABLA DE ANDEVA (ANOVA)

Fte. de va	Gr. de libertad	S. de cuadrados	S. de cuad. medio
Regresion	1.0	19.3	19.3
Residual	12.0	1.5	0.1
Total	13.0	20.8	0.0

TABLA DE ANDEVA (ANOVA)

Fte. de va	F
Regresion	152.3
Residual	0.0
Total	0.0

```
      Programa : PROG44.EST

INICIO

%       TRES PROFESORES DESPUES DE IMPARTIR UN CURSO DE
   INTRODUCCION ALA ESTADISTICA, DECIDEN COMPARAR LAS
   CALIFICACIONES OTORGADAS POR CADA UNO DE ELLOS , PARA
   AVERIGUAR SI EXISTEN DIFERENCIAS SIGNIFICATIVAS EN
   SUS POLITICAS DE CALIFICACION.

   SE DESEA PROBAR LA HIPOTESIS :

     Ho  :  La  probabilidad  de que un  estudiante  obtenga  una
          calificacion  es independiente del profesor con el  que
          curso la materia.

     Ha  :  La  calificacion  que  obtuvo  un  estudiante  no  es
          independiente del profesor que impartio la materia.

   Para realizar la prueba de hipotesis se usa la estadistica  Ji
cuadrada.

   Sea  W95  =18.31 el cuantil de orden .95 para una Ji  cuadrada
con (3-1)*(6-1) = 12 grados de libertad.

-----------------------------------------------------------------;

ARCHIVO LECTURA
                CALIF    = 'CALIFIC.DAT';
ARCHIVO
                LISTADO = 'LP44.DAT';

CONSTANTE
   NCAL    = 7,
   NCAL1   = 8,
   NPROF   = 3,
   NPROF1  = 4,
   W95     = 18.31;

TABLA
   OBSERVADOS [ NPROF1,NCAL1 ],
   ESPERADOS  [ NPROF1,NCAL1 ];

REAL
   JI,TOTAL;

   %   SE LEE LOS DATOS;
   LEE DE CALIF CON '713,/'  ESTO
                    OBSERVADOS.CUERPO[1_NPROF,1_NCAL];

   %   SE CALCULAN LOS TOTALES POR COLUMNA ;
   OBSERVADOS.CUERPO[ NPROF1 , 1_NCAL ]:=
          (OBSERVADOS.CUERPO[ 1_NPROF, 1_NCAL] )[ . , ];
```

```
% SE CALCULAN LOS TOTALES POR RENGLON   ;
OBSERVADOS.CUERPO[1_NPROF1,NCAL1 ]:=
       ( (OBSERVADOS.CUERPO[ 1_NPROF1 , 1_NCAL ])[ , . ] );

% TOTAL DE ALUMNOS;
TOTAL := OBSERVADOS.CUERPO[ NPROF1,NCAL1 ];

% SE CALCULAN LOS VALORES OBSERVADOS    ;
ESPERADOS.CUERPO[ 1_NPROF , 1_NCAL ]:=
((OBSERVADOS.CUERPO[ 1_NPROF,NCAL1 ]) ><
 ((OBSERVADOS.CUERPO[NPROF1,1_NCAL])))/TOTAL;

%  SE  CALCULAN TOTALES POR COLUMNA Y RENGLON DE  LOS  VALORES
  ESPERADOS;
ESPERADOS.CUERPO[ NPROF1 , 1_NCAL ]:=
             (ESPERADOS.CUERPO[ 1_NPROF, 1_NCAL] )[ . , ];
ESPERADOS.CUERPO[1_NPROF1,NCAL1 ]:=
            (ESPERADOS.CUERPO[ 1_NPROF1 , 1_NCAL ])[ , . ];

% SE CALCULA LA ESTADISTICA JI CUADRADA;
JI :=((OBSERVADOS.CUERPO[1_NPROF,1_NCAL]**2)/
   (ESPERADOS.CUERPO[1_NPROF,1_NCAL])) [.,.]  - TOTAL;

% SE LLENA LA TABLA DE VALORES OBSERVADOS;
OBSERVADOS.TITULO := 'Alumnos Observados';
OBSERVADOS.TITREN := 'Profesor';
OBSERVADOS.TITCOL := 'Calificaciones';
OBSERVADOS.NOMREN := [[NPROF1 : 'Perez Lopez','Garcia Garcia',
                      'Lopez Lopez','Tot p cal.'     ]];
OBSERVADOS.NOMCOL := [[ NCAL1 :  ' A ', ' B ', ' C ',' D ',
                       ' E ',' F ',' G ', 'Tot. p. Prof.' ]];
OBSERVADOS.MARCOREN := [[ NPROF1 : NPROF1(3) ]];
OBSERVADOS.MARCOCOL := [[ NCAL1  : NCAL(5.0),8.0 ]];

% SE LLENA LA TABLA DE VALORES ESPERADOS;
ESPERADOS.TITULO := 'Alumnos Esperados';
ESPERADOS.TITREN := 'Profesor';
ESPERADOS.TITCOL := 'Calificaciones';
ESPERADOS.NOMREN := OBSERVADOS.NOMREN;
ESPERADOS.NOMCOL := OBSERVADOS.NOMCOL;
ESPERADOS.MARCOREN := OBSERVADOS.MARCOREN;
ESPERADOS.MARCOCOL := OBSERVADOS.MARCOCOL;

% SE IMPRIMEN LAS TABLAS;

ESCRIBE EN LISTADO CON LIBRE ESTO OBSERVADOS;

ESCRIBE EN LISTADO CON LIBRE ESTO ESPERADOS;

% SE ESCRIBEN EL CUANTIL Y LA ESTADISTICA JI CUADRADA;
ESCRIBE EN LISTADO CON
      '4/,10X,$El valor de la estadiatica Ji es :$,F15.3,/,',
      '10X,$El cuantil de orden .95 es       :$,F15.3,
 ESTO JI,W95;
```

```
   SI JI > W95 ENTONCES
      ESCRIBE EN LISTADO CON
      '10X,$Existe  evidencia  para suponer que  la  calificacion
obtenida por un $,/,',
      '10X,$alumno  no es independiente del profesor con el  cual
curso la materia$,2/,',
      '10X,$Los profesores no califican de igual manera$,4/'
  SINO
     ESCRIBE EN LISTADO CON
     '10X,$Podemos  suponer que los profesores  tienen  identicas
politicas para $,/,',
     '10X,$calificar a los alumnos$,4/';

FIN; % DE PROGRAMA;
```

Archivo de salida del programa : PROG44.EST

Alumnos Observados

Calificaciones

Profesor	A	B	C	D	E	F	G	Tot. p. P
Perez Lope	12	45	49	6	13	18	2	145
Garcia Gar	10	32	43	18	4	12	6	125
Lopez Lope	15	19	32	20	6	9	7	108
Tot p cal.	37	96	124	44	23	39	15	378

Alumnos Esperados

Calificaciones

Profesor	A	B	C	D	E	F	G	Tot. p. P

Perez Lope	14	37	48	17	9	15	6	145
Garcia Gar	12	32	41	15	8	13	5	125
Lopez Lope	11	27	35	13	7	11	4	108
Tot p cal.	37	96	124	44	23	39	15	378

El valor de la estadiatica Ji es : 28.915
El cuantil de orden .95 es : 18.310

Existe evidencia para suponer que la calificacion obtenida por un alumno no es independiente del profesor con el cual curso la mate

Los profesores no califican de igual manera

APENDICE F.
MANUAL DEL COMPILADOR.

En los párrafos siguientes se presenta un breve descripción de la manera en que están organizados los archivos en los discos y un pequeño manual de operación del compilador.

MANUAL DE REFERENCIA.

El compilador se ha implantado en una microcomputadora PC, con 256 K en memoria principal y dos unidades de disco.Se usaron discos flexibles de 5 y 1/4 de pulgada , de doble densidad y doble lado. Cada disco es capaz de almacenar hasta 360 K.

El compilador está escrito en Turbo Pascal (versión 3.0A) y el sistema operativo usado fue MS-DOS (versión 2.11)

El primer disco (Prog. Fuentes) contiene los programas fuente de los programas de apoyo y los analizadores léxico-sintáctico y semántico. Un segundo disco (Objetos) debe contener a los programa objeto de los analizadores y en un tercer disco (Biblioteca) están alamacenados los archivos que constituyen a las rutinas de biblioteca.

DISCO 1

ARCHIVO	CONTENIDO
COMMAND.COM	Sistema operativo
GTO.EST	Gramática del lenguaje
PALCL8.EST	Palabras reservadas del lenguaje
HT7.PAS	Programa que construye la Tabla Gramatical
HPC2.PAS	Programa que construye el diccionario de palabras reservadas
LEXICO.PAS	Programa fuente del analizador Léxico-Sintáctico

Los siguientes archivos constituyen al analizador semántico y también viven en el disco 1:

ARCHIVO	CONTENIDO
INIFYIN.PAS	Compila inicio y fin de un nivel lexicográfico
GLOBAL.PAS	Rutinas usadas en todo momento de compilación
DECLARA.PAS	Rutinas que declaran a los distintos objetos declarables en el lenguaje
EXPR1.PAS EXPR2.PAS	Rutinas que compilan a una expresión aritmética
ESTC1.PAS ESTC2.PAS	Rutinas que compilan a las estructuras de control
GOTOYCON.PAS	Rutina que procesa la primera parte de la proposición condicional
RUTINAS.PAS	Rutinas que compilan el llamado a una rutina sin tipo
LEE.PAS	Rutinas que procesan a las proposiciones de entrada-salida

DISCO 2

ARCHIVO	CONTENIDO
COMMAND.COM	Sistema operativo
TURBO3.COM	Compilador de PASCAL y editor
TURBO.MSG	Mensajes del compilador de PASCAL
GRAMGEN	Tabla Gramatical
METASIMB	Diccionario de símbolos no terminales
PCCHAR PCVBT	Diccionario de palabras clave
LEXICO.COM	Analizador sintáctico
SEM12.COM	Analizador semántico

DISCO 3

ARCHIVO	CONTENIDO
COMMAND.COM	Sistema operativo
UCOD0.PAS	Declara a las variables usadas en las rutinas de biblioteca
INICI.PAS	Inicializa a las variables declaradas en UCOD0.PAS
UCOD1.PAS	Rutinas generales
UCOD2.PAS	Operaciones unarias y binarias Manejo de arreglos explícitos
UCOD3.PAS	Manejo de expresion selectiva de arreglos Rutinas de asignación Rutinas usadas por las estructuras de control
UCOD5.PAS	Código común a proposiciones de entrada y salida
UCOD6.PAS	Código exclusivo de la proposición LEE
UCOD7.PAS	Código exclusivo de la proposición ESCRIBE
UCOD8.PAS	Rutinas para la impresión de Tablas, Gráficas e Histogramas Rutinas para la construcción de Gráficas

Algunos archivos tienen una terminación numérica, como por ejemplo SEM12.COM. Por cada nueva versión del archivo debe actualizarse la terminación numérica.

MANUAL DE OPERACION

Los pasos a seguir para crear un programa en el lenguaje estadistico son :

-- Colocar el Disco 2 en el drive B y el disco 3 en el drive A
-- Encender la máquina y esperar a que esta cargue el sistema operativo, proporcionando fecha y hora
-- Instalado el sistema operativo, aparece en la pantalla :
A>_
-- Se teclea : B: (return)
y debe aparecer : B>_

En este momento se puede iniciar la edición o compilación de un programa.

EDICION.

-- Teclear : TURBO3 (return)
-- A la pregunta : Include messages errors?
contestar : N
-- Invocar al editor con el comando W, y proporcionar el nombre del archivo a editar. El nombre del archivo debe tener la extensión .EST
-- Se edita el archivo
-- Termina la edición con el comando : CTRL K / D
-- Se salva el archivo con el comando S
-- Se sale de Turbo-Pascal con el comando Q

El manejo del editor de Turbo-Pascal puede consultarse en [T1].

COMPILACION

-- Anotar : LEXICO (return)
-- Proporcionar el nombre del archivo a compilar
-- Proporcionar el dispositivo de salida, que puede ser :
--- Archivo en disco
--- Pantalla : CON:
--- Impresora : LST:
-- Si el analizador léxico-sintáctico encuentra un error, corregir el programa fuente a través del proceso de edición. Cuando no exista un error, continuar con los siguientes pasos.
-- Anotar : SEM12 (return)
-- Proporcionar el dispositivo de salida, que puede ser un archivo en disco, la terminal de video o la impresora
-- Proporcionar el dispositivo en el cuál se grabará el código generado, este puede ser :
--- Un archivo en disco con extensión .PAS
--- La terminal de video o la impresora
-- Si fueron encontrados errores debe corregirse el programa fuente, usando el proceso de edición; cuando no se encontraron errores puede obtenerse el código ejecutable.

EJECUCION

-- Si el sistema operativo está instalado (aparece: B>), son repetidos los dos primeros pasos del proceso de edición
-- Es invocado el compilador de PASCAL con el comando C y se proporciona el nombre del archivo a compilar
-- Si no existieron errores detectados por el compilador de PASCAL, se puede ejecutar el programa con el comando R .
-- También puede salvarse el código ejecutable con la siguiente serie de comandos :
---- O Opciones de Compilación
---- C Crear archivo .COM
---- Q Salir de 'Opciones de compilación '
---- C Compilar el archivo con extension .COM

Los archivos con extensión .COM, deben ser invocados desde el sistema operativo .

La capacidad del compilador puede ser sobrepasada cuando se compilan programas grandes. La solución consiste en extender el uso de OVERLAYS a otras rutinas de biblioteca.

Una explicación del manejo de OVERLAY en Turbo-Pascal puede leerse en [T1].

Los tamaños de los programas objeto son :

Archivo	Tamaño
LEXICO.COM	19256 bytes
SEM12.COM	58783 bytes
UCOD#.COM	48928 bytes

El archivo UCOD#.COM no existe en los discos, pero es el tamaño del código correspondiente a todas las rutinas de biblioteca.

Porque las rutinas de biblioteca no pueden ser ligadas, éstas se deben compilar junto con el programa fuente, usando mucho la lectura de disco. Por esta razón se recomienda que el compilador se instale en maquinas con disco duro.

BIBLIOGRAFIA.

B1. Barrow, David; Pascal : The Lenguage and its Implementation; John Wiley & Sons; 1981;

E1. Esquivel Avila, Carlos; Desarrollo de un Compilador para un Lenguaje Orientado a Resolver Problemas EStadisticos; Tesis de Licenciatura de Actuario, UNAM; 1986.

F1. Forsythe, George E.; Malcolm, Michael A. y Moler, Cleve. B; Computer Methods for Mathematical Computations; Prentice-Hall.

G1. García García, Javier ; Diseño de un Lenguaje Orientado a Resolver Problemas Estadisticos; Tesis de Licenciatura de Actuario, UNAM; 1983.

G2. Gries, David; Compiler Construction for Digital Computers; John Wiley & Sons; 1971.

L1. Legarreta García , Luis; Compiladores; Fundación Arturo Rosemblueth; México, D.F.; 1983.

T1. Turbo Pascal Reference Manual V3.0 Scotts Valley , Borland International; 1985.

K1. Kleimbaum, G. David; Applied Regression and other Multivariable Methods ;Duxbury Press ; 1978.

W1. Waite, William M. y Goos, Gerhard; Compiler Construction; Springer-Verlag.