UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Facultad de Ciencias

Diseño de un Lenguaje Orientado a Resolver Problemas Estadísticos

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 :

Javier García García

MEXICO, D. F. 1983

INDICE

APENDICES

1.OBJETIVO.

La idea de realizar el presente trabajo surgiò de lo siguiente. El grupo de estadistica de la Facultad de Ciencias se encuentra muy ligado a la investigaciòn y enseñanza de la estadistica y a èl recurren diversas instituciones para la resoluciòn de problemas de muy diversa indole relacionados con campos de estudio como Biologia, Medicina, Ciencias Quimicas, Ciencias Sociales, Ciencias de la Atmòsfera entre otros.

Se cuenta con un laboratorio el cual tiene una minicomputadora NOVA 3/12 cuyas caracteristicas se encuentran en el apèndice A, la cual no es usada a su màxima capacidad; existe el equipo pero no se tiene la infraestructura suficiente para poder aprovecharlo plenamente. Asi, los problemas tienen que resolverse casi ad hoc y resulta que cada vez que surge un nuevo problema es dificil aprovechar lo hecho anteriormente y pràcticamente se inicia de cero. Asi pues, se pensò que estaban dadas las condiciones para la elaboraciòn de un lenguaje computacional orientado a resolver problemas estadisticos. El diseño de dicho lenguaje y el desarrollo de un compilador para el mismo constituyen el tema central de esta tesis.

Es importante hacer notar que muchas decisiones se tomaron observando que se deseaba que la nueva herramienta se adaptase al equipo existente en el laboratorio. Por ejemplo la decisiòn de que el compilador fuese diseñado para generar còdigo Fortran y que el propio compilador esta escrito tambièn en este lenguaje fueron dos de ellas.

En plàticas celebradas con el grupo de estadistica primeramente, en el campo de la enseñanza, se nos planteò lo siguiente : En los primeros cursos de estadistica los alumnos se enfrentan a problemas con cierta complejidad numèrica en los que tienen que trabajar con volùmenes de informaciòn de un tamaño tal que seria casi imposible manejarlos manualmente. Esto provoca que tengan que hacer uso de una computadora en una materia en que no es prerequisito haber llevado cursos de còmputo. El alumno, al enfrentarse a este problema tiene dos opciones: Por un lado perderse en un bosque tratando de elaborar programas especificos y complicados en un lenguaje extraño perdiendo muchas veces el panorama general del problema que desea resolver, lo que provoca que se forme una visiòn errònea de lo que es realmente la materia. Por otro lado, utilizar un paquete estadistico con el cual ùnicamente sigue una serie de pasos que rara vez entiende pero sabe

que lo llevan al resultado deseado, sin haber aplicado la teoria que durante todo el curso se estuvo analizando, llegando al extremo de pensar que lo que se encuentra en el paquete es lo ùnico realizable. Los diseñadores de estas herramientas, con el objeto de facilitar su uso, hacen que los algoritmos utilizados en los càlculos sean transparentes al usuario.

Decidièndose por la primera opciòn, el grupo de estadistica, con el objeto de resolver en parte este problema, elaborò subrutinas para que el alumno las integrase a sus programas evitàndole asi un poco el trabajo pesado. Se notò que muchas de las subrutinas que se hicieron tienen que ver con el manejo de matrices (lectura y escritura de matrices, obtencion de submatrices a partir de una matriz dada, obtencion de inversas, traspuestas, productos, etc.).

Pero el problema aun persiste pues la notaciòn que se usa es muy distinta a la notaciòn utilizada en el campo de la estadistica.

Por otro lado, ya no en el campo de la enseñanza sino en el de la investigaciòn y la aplicaciòn de la estadistica existen varios problemas.

El aprender un lenguaje computacional se vuelve una necesidad, lo cual implica ya una dificultad pues los problemas se plantean en un lenguaje ajeno al campo.

El utilizar paquetes estadisticos tiene la ventaja de que su manejo es fàcil pero sus posibilidades son fijas y dificilmente modificables lo cual presenta una gran limitaciòn en un campo en el que surgen constantemente nuevos mètodos.

De lo planteado anteriormente, salta a la vista la necesidad de diseñar un lenguaje computacional especialmente orientado a resolver problemas estadisticos, que contemple una notaciòn afin a la utilizada en la disciplina, facilitando de esta manera su aprendizaje y la lectura de los algoritmos escritos en el lenguaje. Debe ser lo suficientemente general para poder expresar con èl gran variedad de problemas.

Por otro lado, pensando en que la gramàtica del lenguaje fuese fàcilmente modificable durante su desarrollo y que el compilador debiera ser transportable y con una estructura tal que permitiera su implementaciòn en una minicomputadora, se pensò en desarrollar un compilador guiado por tablas.

En las siguientes lìneas se plantearà el diseño del

lenguaje marcàndose primeramente las ideas principales del mismo. A continuaciòn se describirà el compilador.

2. ASPECTOS FUNDAMENTALES DEL DISEÑO DEL LENGUAJE

2.1 Introducción.

Para diseñar el lenguaje se tomó muy en cuenta la opinión de los futuros usuarios principalmente estudiantes de los cursos de estadística, profesores de dichos cursos, investigadores del campo. Con el objeto de cubrir lo más que fuese posible los problemas a los que se pudiesen enfrentar, así como para estudiar la notación utilizada, se decidió revisar los temas y la bibliografía de los cursos de estadística como son: Estadística I (que incluye temas como estimación, intervalos de confianza, pruebas de hipótesis), Estadística II (que incluye temas como estadística no paramétrica, prueba de k-muestras, correlación), Análisis de Regresión, Estadística Bayesiana, Diseño de Experimentos, Muestreo, Métodos Multivariados y diversos seminarios.

Se estudiaron diversos lenguajes computacionales al igual que paquetes estadísticos para analizar su notación, así como la clase de problemas para los cuales fueron diseñados. En la descripción de nuestro lenguaje se mencionarán las fuentes de donde se tomaron las ideas de diseño. Al respecto quisieramos citar una idea interesante expresada por N. Wirth [J1]

> *"Es claro que un nuevo lenguaje no debe ser desarrollado sólo por presentar algo nuevo; los lenguajes existentes deben ser usados como base de desarrollo..."*

El resultado de este estudio fue un lenguaje cuyas características relevantes son las siguientes:

Engloba en una forma coherente estructuras de datos y de control formando una unidad homogénea.

El haber inspirado su estructura en lenguajes como Pascal y Algol hace que herede de estos sus principios de estructuración. Por lo tanto no se limita a unir elementos que faciliten la construcción de algoritmos del campo de la estadística. Constituye un lenguaje de propósito general que en su diseño plasma los conceptos de programación estructurada. Se dió importancia a que la notación fuese afín a la utilizada en la disciplina estadística y las estructuras presentadas constituyen las más comunmente utilizadas en el campo, sin perder su generalidad. Así mismo en las operaciones sobre estas estructuras se trató de eliminar restricciones arbitrarias buscando así la ortogonalidad del lenguaje.

[T1]. El concepto de ortogonalidad supone que en general las reglas se aplican en todos los contextos. Por ejemplo, siempre que sea vàlido el uso de una constante se puede poner una expresiòn aritmètica.

Antes de describir el lenguaje resumiremos en las siguientes lìneas los aspectos fundamentales. Dividiremos nuestro resumen en: Tipos y Declaraciones, Expresiones y Proposiciones.

2.2 Tipos y Declaraciones

Los programas tienen una estructura de bloques. En cada bloque existe una descripciòn de las acciones a realizar y una definiciòn de las estructuras que se manipulan en las acciones. En las declaraciones, las propiedades de las estructuras quedan definidas por el tipo al cual se les asocia y se les da un identificador mediante el cual puede uno referirse a ellas. La declaraciòn de un identificador es vàlida en el bloque donde se encuentra.

En nuestro lenguaje hay dos tipos bàsicos: Real y Texto. Existen tambièn tipos estructurados: Arreglos, Gràficas, Tablas e Histogramas; existe un tipo adicional contenido en el tipo Gràfica llamado Cuerpo Gràfica. Para las operaciones de entrada y salida se incluyeron dos tipos màs: Formatos y Archivos.

Los valores denotados por el tipo Real incluyen los tipos 'real' e 'integer' de Pascal. El no hacer distinciòn entre nùmeros de punto fijo y punto flotante se tomò del lenguaje APL y de las versiones originales de Basic. Esto se hizo con el objeto de facilitar el manejo de variables numèricas en general.

Las estructuras tipo Arreglo juegan un papel relevante en el lenguaje. Pensemos simplemente en la importancia de estas estructuras especialmente las matrices en el campo de la estadistica. Su importancia queda demostrada ùnicamente pensando en que "...la forma màs simple de procesar datos es formar un arreglo con la informaciòn.."[CH1].

Un tipo Arreglo consta de componentes de un solo tipo bàsico. En la declaraciòn de estas estructuras se especifica el nùmero de dimensiones al igual que el nùmero de elementos por dimensiòn; asi mismo se indica el tipo bàsico asociado al arreglo. En el caso de los arreglos de una dimensiòn de tipo Real se puede especificar si el arreglo es renglòn o columna para las operaciones en que se necesite que los operandos sean conformables. Es importante hacer notar que el nùmero de

dimensiones es parte del tipo mientras que los límites no lo son.

Por otro lado en cuanto a los tipos Gràfica, Tabla, e Histograma se vió la importancia de su creación por el papel que juegan estas estructuras tanto para describir como para presentar información estadística.

Para su elaboración se tomaron ideas de diversos paquetes estadísticos, principalmente el SPSS y el paquete BASIS que de alguna forma u otra atacan el problema de descripción y presentación de información estadística.

A diferencia de los arreglos, estos tipos estructurados son heterogèneos en el sentido de que mezclan tipos en su estructura.

Bajo un identificador se engloba una estructura que contiene diversos campos que incluyen títulos para identificar la estructura, subtítulos, nombres de renglones y columnas, nombres de escalas y frecuencias, etc.

Más adelante se explicará cada uno de estos campos y como la proposición de escritura los distribuye en el archivo de salida.

Los tipos Formato y Archivo fueron creados para ser utilizados en las proposiciones de entrada y salida. En la declaración de formatos se asocia un identificador a una serie de caracteres de edición. Por facilidad se eligió que estos caracteres fuesen iguales a los del lenguaje Fortran [D3]. En la declaración de archivos se asocia un identificador a un archivo secuencial. El archivo se abrirá al inicio del programa y se cerrará al final del mismo.

Por último se vió la importancia de poder declarar rutinas. Estas estan inspiradas en los procedimientos y funciones de Pascal.

2.3 Expresiones

Las expresiones juegan un papel muy importante en el lenguaje. La necesidad en el campo de la estadística de poder manejar cálculos de una manera fàcil para que estos no sean obstàculo al tratar de dominar los mètodos estadísticos es evidente. Al respecto podemos citar lo siguiente:[S1]

"El impacto de la Teoría Estadística Moderna ya se conocía por el año de 1930,

pero era escasamente usada. Esto se debió a que la acumulación y análisis de datos estadísticos, aun con la ayuda de una teoría poderosa, involucraba cálculos tediosos y pesados...es innecesario estudiar los trucos y atajos que existen en los cálculos manuales que recientemente ocupan mucho del curriculum estadístico. Ahora la necesidad es dominar los métodos estadísticos"

En las expresiones se combinan operandos y operadores. Esta combinación está provista de significado de tal forma que se obtienen nuevos valores al interpretar o evaluar la expresión.

En el lenguaje se define un conjunto fijo de operadores cada uno de los cuales representa una función cuyo dominio lo forman los tipos de sus operandos y su rango el tipo del resultado.

Al evaluarse una expresión el valor de esta caerá en algún tipo que puede ser cualquiera de los básicos y estructurados mencionados anteriormente.

Se observó la importancia de utilizar operadores en estructuras como: vectores, matrices y en general estructuras de tipo Arreglo Real n-dimensional. Se incluyó una notación para poder expresar constantes de tipo Arreglo n-dimensional ya sea de tipo Real o Texto en las expresiones. Esta notación está inspirada en una análoga en el lenguaje APL [I1].

Se incluyeron operadores sobre operandos de tipo Arreglo Real de dos dimensiones, como el operador "><" (producto de matrices), "'" (traspuesta), "@¹" (inversa).

Se propuso también una notación para obtener estructuras de tipo Arreglo m-dimensional a partir de otras estructuras de tipo Arreglo n-dimensional Real o Texto, con m igual o menor a n, pudiendose seleccionar sólo ciertos elementos o rangos de elementos de cada dimensión. Esta herramienta resulta muy útil para la obtención de submatrices. Una notación similar existe en APL [I1].

Tomando en cuenta que es muy generalizado el tener que calcular la suma de uno o más renglones, columnas, planos, etc. de una estructura de tipo Arreglo Real n-dimensional, se agregó una notación para realizar estas operaciones.

En el resultado de estas expresiones el número de

dimensiones podría quedar indefinido. Por ejemplo es concebible visualizar un vector como una matriz de un solo renglón; particularmente importante resulta considerar esto al efectuar una asignación o en el caso en que el número de dimensiones sea igual a dos debido a la frecuencia del uso de estas estructuras y los operadores definidos sobre ellas. Para esto, antes de aplicarse algún operador sobre expresiones de cualquier tipo Arreglo Real, estas se "dimensionan" o sea que se calculan sus dimensiones reales con el objeto de ver si es posible aplicar tal o cual operador. Esta operación esta inspirada en las "coerciones" de Algol 68 [T1]. Las reglas que se siguen para dimensionar una expresión se verán más adelante.

Se propuso el poder efectuar operaciones sobre estructuras de tipo Cuerpo Gráfica, el cual es un tipo asociado a la estructura Gráfica pues surge frecuentemente la necesidad de comparar en un mismo plano dos o más gráficas.

Los operadores de relación definidos en el lenguaje (<=, <, <>, >=, >) así como los operadores lógicos (Y, O y NO) al aplicarse dan como resultado expresiones de tipo Real o Arreglo Real de n-dimensiones con unos y ceros, sustituyendo asi las expresiones de tipo booleano siguiendo en cierto sentido la forma como se manejan en APL [I1].

En las expresiones se pueden utilizar dos constantes tipo Real predefinidas donde se requieran: PI, E. Donde los valores de PI y E son muy cercanos a las constantes matemáticas.

Por último, entre los operandos de una expresión se puede incluir el llamado a una rutina con tipo. Esta clase de rutinas entrega un valor de tipo Real.

2.4 Proposiciones

La proposición más importante del lenguaje es la asignación. Del lado derecho del símbolo de asignación (:=) debe haber una expresión y del lado izquierdo un objeto, ya sea una variable o una estructura congruente con el tipo del resultado de la expresión. Al ejecutarse la asignación, el valor de este último objeto es reemplazado por el valor de la expresión. Existe otro tipo de asignación que se verá en las rutinas con tipo.

En el lenguaje se incluyeron también proposiciones de entrada y salida que operan sobre archivos

secuenciales. Diversos paquetes estadísticos fueron analizados para diseñar estas proposiciones. Varias ideas fueron tomadas principalmente del paquete SPSS [N2].

De la proposición de lectura cabe resaltar lo siguiente. Existe la posibilidad de especificar el formato en que vienen los datos de entrada, los cuales pueden ser numéricos o alfanuméricos, pudiéndose definir todos los campos o sólo los de interés.

En muchas aplicaciones estadísticas la información contenida en un archivo puede verse como una lista de casos donde cada caso contiene N campos que ocupan uno o más registros en el archivo. Así pues la información se puede ver como una matriz donde las columnas son los campos que conforman cada caso y los renglones, que físicamente pueden ocupar uno o más registros, son los casos.

Existe la posibilidad de asignar valores a objetos de tipo Real o Texto, o Arreglo Real o Texto de una dimensión mediante la lectura especificándose en el formato los campos y el número de registros que ocupa cada caso.

En la proposición de escritura es posible especificar el formato de salida al escribir expresiones de tipo Real, Texto o Arreglo de una dimensión. Por otro lado, la proposición de escritura "sabe" distribuir los campos de las estructuras Tabla, Gráfica e Histograma como se indicará más adelante.

Otra proposición del lenguaje es el llamado a una rutina sin tipo la cual provoca la ejecución de una rutina previamente declarada. Esta proposición es similar al llamado a un procedimiento en Pascal [J1].

En las dos clases de rutinas (con y sin tipo) el paso de parámetros de tipo Real puede ser por valor o por referencia. El paso de los demás parámetros es por referencia.

En la declaración de la rutina puede a su vez haber declaraciones las cuales serán locales a ella.

Hasta este punto no hemos mencionado nada acerca del flujo de control de los programas. La ejecución de las proposiciones es como la de Algol y Pascal. Existen los conceptos de bloque y proposición compuesta los cuales especifican la ejecución secuencial.

En el lenguaje existen proposiciones para cambiar el flujo secuencial de ejecución.

La ejecución condicional o selectiva puede ser especificada mediante las proposiciones SI..ENTONCES, SI..ENTONCES..SINO, LA..DE las cuales son traducciones de las proposiciones IF..THEN, IF..THEN..ELSE, CASE..OF de Algol respectivamente [N1].

La ejecución repetitiva de proposiciones se especifica con la proposición REPITE cuyas variantes incluyen proposiciones similares al WHILE, DO..UNTIL, FOR..STEP..UNTIL de Algol.

Para decidir que acción tomar, estas proposiciones evalúan las expresiones de tipo Real correspondientes. Si las expresiones no son iguales a cero, esto provocará que se tome la acción correspondiente al resultado verdadero de Algol, en caso contrario se elegirá la alternativa.

Por último existe un equivalente al GO TO de Pascal: la proposición VER.

3. EL LENGUAJE

Dividiremos nuestra descripciòn del lenguaje en cuatro partes las cuales son:

3.1) Componentes del lenguaje
3.2) Declaraciones
3.3) Expresiones
3.4) Proposiciones

La sintaxis libre de contexto se definirà utilizando BNF agregàndose los simbolos (* y *) los cuales se utilizaràn para describir algùn simbolo de la gramàtica que sea impràctico o imposible definir con una fòrmula. El simbolo "!" se utilizarà para representar la conjunciòn O de BNF por la falta del simbolo correspondiente.

Se darà una explicaciòn para aclarar la semàntica del lenguaje.

3.1 Componentes del lenguaje

3.1.1 Simbolos

Los simbolos del lenguaje seràn las letras, los digitos, simbolos especiales, palabras reservadas y el vacio

```
<SIMBOLO>::=<LETRA>!<DIGITO>!<SIMBOLO ESPECIAL>!
     <PALABRA RESERVADA>!<VACIO>
<LETRA>::=(*DE LA LETRA A A LA Z*)
<DIGITO>::=(*NUMEROS ENTEROS DEL CERO AL NUEVE*)
<SIMBOLO ESPECIAL>::= + ! - ! * ! / ! <= ! < !
     = ! <> ! >= ! > ! _ ! & ! ( ! ) !
     ' ! @ ! ; ! . ! : ! >< ! % !
     [ ! [[ ! ] ! ]] ! ** ! , ! " ! := !
     (* SIMBOLO ! VER NOTA*)
<PALABRA RESERVADA>::= INICIO ! FIN ! CONSTANTE !
     REAL ! TEXTO ! ARREGLO ! SI ! SINO !
     ENTONCES ! DE ! CON ! LA ! DE ! REPITE !
     HASTA ! SUMANDO ! MIENTRAS ! VER !
     RUTINA ! VAL ! NO ! Y ! O ! PI ! E !
     ESCRIBE ! EN ! LIBRE ! ESTO ! LEE !
     REN ! COL ! GRAFICA ! TABLA ! HISTOGRAMA !
     TITULO ! TITREN ! TITCOL ! NOMREN !
     NOMCOL ! MARCOREN ! MARCOCOL ! CUERPO !
     ARCHIVO ! FORMATO
<VACIO>::=(*AUSENCIA TOTAL DE SIMBOLOS*)
```

Nota: Se vió la necesidad de utilizar el simbolo "!" para indicar el càlculo de un determinante siendo èste el ùnico caso en que aparecerà como un simbolo terminal. En la

descripciòn de la gramàtica se aclararà debidamente para evitar confuciones.

Dado que no en todas las màquinas existen este tipo de simbolos, estos se podràn cambiar haciendo las aclaraciones pertinentes.

3.1.2 Identificadores

Los identificadores no tienen un significado intrinseco y sirven para denotar constantes, variables. estructuras, etc. Como ya se mencionò, la declaraciòn de un identificador es vàlida en el bloque donde se encuentra, fuera de èste puede utilizarse para otros propòsitos.

```
<IDENTIFICADOR>::=<LETRA><RESTO IDENTIFICADOR>
<RESTO IDENTIFICADOR>::=<LETRA>
     <RESTO IDENTIFICADOR> !
     <DIGITO><RESTO IDENTIFICADOR> !
     <VACIO>
```

3.1.3 Nùmeros

Los nùmeros conforman los valores del tipo bàsico Real. Como ya se ha dicho, incluyen los tipos 'real' e 'integer' de Pascal.

```
<NUMERO>::=<NUMERO DECIMAL><PARTE EXPONENCIAL>
<NUMERO DECIMAL>::=<NUMERO ENTERO> !
     <NUMERO ENTERO>.<PARTE DECIMAL>
<NUMERO ENTERO>::=(*SECUENCIA DE UNO O MAS DIGITOS*)
<PARTE DECIMAL>::=(*SECUENCIA DE UNO O MAS DIGITOS*)
<PARTE EXPONENCIAL>::=E<SIGNO><NUMERO ENTERO>!
     <VACIO>
<SIGNO>::= + ! - ! <VACIO>
```

3.1.4 Textos

Los textos conforman los valores del tipo bàsico Texto. Estos textos pueden ser de a lo màs 60 caracteres.

```
<TEXTO>::="<CADENA DE SIMBOLOS>"
<CADENA DE SIMBOLOS>::=(*SECUENCIA DE
     CERO O MAS SIMBOLOS*)
```

Nota. Si en la cadena de simbolos deben incluirse comillas, estas se repetiràn dos veces.

3.1.5 Comentarios

Pueden insertarse comentarios a lo largo del

programa entre identificadores, palabras reservadas, números, textos y símbolos especiales

```
<COMENTARIO>::=%<CADENA DE SIMBOLOS>;
```

Nota. La cadena de símbolos no debe incluir un punto y coma (;) pues este símbolo termina el comentario.

3.1.6 Programas

Un algoritmo expresado en el lenguaje constituye un programa.

```
<PROGRAMA>::=<BLOQUE>; ! <PROPOSICION COMPUESTA>;
<BLOQUE>::=<INICIO DE BLOQUE>;<FIN DE BLOQUE>
<INICIO DE BLOQUE>::=INICIO
     <LISTA DE DECLARACIONES>
<LISTA DE DECLARACIONES>::=<DECLARACION> !
     <DECLARACION>;<LISTA DE DECLARACIONES>
<FIN DE BLOQUE>::=<PROPOSICION>FIN !
     <PROPOSICION>;<FIN DE BLOQUE>
<PROPOSICION COMPUESTA>::=INICIO<FIN DE BLOQUE>
<DECLARACION>::=<DECLARACION DE CONSTANTE> !
     <DECLARACION DE REAL> !
     <DECLARACION DE TEXTO> !
     <DECLARACION DE ARREGLO> !
     <DECLARACION DE GRAFICA> !
     <DECLARACION DE TABLA> !
     <DECLARACION DE HISTOGRAMA> !
     <DECLARACION DE ARCHIVO> !
     <DECLARACION DE RUTINA> !
     <DECLARACION DE FORMATO>
<PROPOSICION>::=<ETIQUETA>:
     <PROPOSICION SIN ETIQUETA> !
     <PROPOSICION SIN ETIQUETA>
<ETIQUETA>::=<NUMERO ENTERO>
<PROPOSICION SIN ETIQUETA>::=
     <PROPOSICION INCONDICIONAL> !
     <PROPOSICION CON CONDICION>
<PROPOSICION INCONDICIONAL>::=
     <PROPOSICION DE ASIGNACION>!
     <PROPOSICION DE LLAMADO A RUTINA>!
     <PROPOSICION DE LECTURA>!
     <PROPOSICION DE ESCRITURA>!
     <PROPOSICION VER>!
     <PROPOSICION COMPUESTA>!
     <PROPOSICION VACIA>
<PROPOSICION CON CONDICION>::=
     <PROPOSICION CONDICIONAL>!
     <PROPOSICION SELECTIVA>!
     <PROPOSICION ITERATIVA>
```

En el apèndice D se encuentran ejemplos de programas donde se ilustra la sintaxis de <NUMERO>, <TEXTO>, <COMENTARIO>, y <PROGRAMA>

3.2 Declaraciones

3.2.1 Declaraciòn de Constante

Con esta declaraciòn asociamos un identificador a un valor de cualquierà de los dos tipos bàsicos.

```
<DECLARACION DE CONSTANTE>::=CONSTANTE
     <LISTA DE CONSTANTES>
<LISTA DE CONSTANTES>::=<DEFINICION DE CONSTANTE>!
     <DEFINICION DE CONSTANTE>,
     <LISTA DE CONSTANTES>
<DEFINICION DE CONSTANTE>::=<IDENTIFICADOR>=
     <NUMERO O TEXTO>
<NUMERO O TEXTO>::=<SIGNO><NUMERO O CONSTANTE> !
     <TEXTO>
<NUMERO O CONSTANTE>::=<NUMERO> !
     <IDENTIFICADOR DE CONSTANTE>
<IDENTIFICADOR DE CONSTANTE>::=<IDENTIFICADOR>
```

Una vez declarada puede ser usada en las declaraciones que le sigan

Ejemplo:

```
CONSTANTE
   MUESTRA = 30,
   ERROR   = 0.03;
```

3.2.2 Declaraciòn de Real y Texto

Declaramos variables de tipo Real y Texto

```
<DECLARACION DE REAL>::=REAL
     <LISTA DE IDENTIFICADORES>
<DECLARACION DE TEXTO>::=TEXTO
     <LISTA DE IDENTIFICADORES>
<LISTA DE IDENTIFICADORES>::=<IDENTIFICADOR>!
     <IDENTIFICADOR>,
     <LISTA DE IDENTIFICADORES>
```

Ejemplo:

```
REAL
  PORCENTAJE, ESTADISTICA;

TEXTO
```

NOMBRE;

El valor inicial de las variables de tipo Real será cero y para las variables de tipo Texto será la cadena de caracteres de longitud cero.

3.2.3 Declaración de Arreglo

Un arreglo es una estructura compuesta de un número fijo de componentes del mismo tipo el cual puede ser Real o Texto. Si no se especifica en la declaración, por omisión el tipo será Real. Estas estructuras se componen de un número fijo pero arbitrario de dimensiones. Si el arreglo es Real de una dimensión debe especificarse si es renglón o columna. Por omisión será columna.

```
<DECLARACION DE ARREGLO>::=ARREGLO
     <TIPO ARREGLO><LISTA DE ARREGLOS>
<TIPO ARREGLO>::=REAL <RENGLON COLUMNA> !
     TEXTO ! <RENGLON COLUMNA>
<RENGLON COLUMNA>::=REN ! COL ! <VACIO>
<LISTA DE ARREGLOS>::=<DEFINICION DE ARREGLO>!
     <DEFINICION DE ARREGLO>,
     <LISTA DE ARREGLOS>
<DEFINICION DE ARREGLO>::=
     <LISTA DE IDENTIFICADORES>
     [<DIMENSIONES>]
<DIMENSIONES>::=<LIMITES>!<LIMITES>,<DIMENSIONES>
<LIMITES>::=<SIGNO><NUMERO O CONSTANTE>:
     <SIGNO><NUMERO O CONSTANTE>
```

Nota. Si los valores de <LIMITES> no se evalúan a números enteros, se redondean. En los <LIMITES>, el primero será menor o igual que el segundo.

Ejemplo:

```
ARREGLO
  A,B[1:MUESTRA,1:5],
  C[1:MUESTRA];

ARREGLO TEXTO
  T[1:10];

ARREGLO REN
  X[1:5];
```

A y B son dos arreglos de tipo Arreglo Real de dos dimensiones, C es de tipo Arreglo Real Columna y X es de tipo Arreglo Real Renglón; T es de tipo Arreglo Texto de

una dimensiòn.

3.2.4 Declaraciòn de Gràfica, Tabla e Histograma

Estas estructuras se componen de un nùmero fijo de elementos de diversos tipos, como se mencionò en la secciòn 2.2. La forma como se declaran es la siguiente:

```
<DECLARACION DE GRAFICA>::=GRAFICA
     <LISTA DE ESTRUCTURAS>
<LISTA DE ESTRUCTURAS>::=
     <DEFINICION DE ESTRUCTURA>!
     <DEFINICION DE ESTRUCTURA>,
     <LISTA DE ESTRUCTURAS>
<DEFINICION DE ESTRUCTURA>::=
     <LISTA DE IDENTIFICADORES>
     [ <SIGNO><NUMERO O CONSTANTE>,
       <SIGNO><NUMERO O CONSTANTE> ]
<DECLARACION DE TABLA>::= TABLA
     <LISTA DE ESTRUCTURAS>
<DECLARACION DE HISTOGRAMA>::= HISTOGRAMA
     <LISTA DE HISTOGRAMAS>
<LISTA DE HISTOGRAMAS>::=
     <DEFINICION DE HISTOGRAMA>!
     <DEFINICION DE HISTOGRAMA>,
     <LISTA DE HISTOGRAMAS>
<DEFINICION DE HISTOGRAMA>::=
     <LISTA DE IDENTIFICADORES>
     [<SIGNO><NUMERO O CONSTANTE>]
```

Ejemplo:

```
GRAFICA
     G1,G2[20,30];
TABLA
     T1[6,5];
HISTOGRAMA
     H1[5];
```

En el ejemplo G1 y G2 son dos gràficas que cuentan con 20 unidades en el eje vertical y 30 en el horizontal. T1 es una tabla de 6 renglones por 5 columnas. Por otro lado H1 es un histograma que contempla 5 frecuencias. Nòtese que <SIGNO><NUMERO O CONSTANTE> deberà ser mayor o igual a uno. Estas estructuras engloban una serie de campos que a continuaciòn se describen

Para referirse a uno de estos campos se utiliza tanto el identificador de la estructura como el nombre asociado al campo separados por un punto siguiendo el mecanismo propuesto por Hoare para el manejo de 'records' [H2].

En el diseño de estas estructuras se contempló el poderlas identificar mediante un titulo (TITULO es el nombre asociado al campo). En el caso del tipo Tabla se incluyeron campos para denotar subtitulos (TITREN y TITCOL) tanto para los nombres de los renglones (NOMREN) como para los de las columnas (NOMCOL) y el poder enmarcar la información (con los campos denotados como MARCOCOL y MARCOREN) la cual la constituye el campo CUERPO. Obsèrvese la figura 3.1

FIGURA 3.1

T I T U L O

	TITCOL		
TITFIL	NOMCOL (1)	NOMCOL (2)	NOMCOL (n)
NOMFIL (1)	CUERPO (1,1)	CUERPO (1,2)	CUERPO (1,n)
NOMFIL (2)	CUERPO (2,1)	CUERPO (2,2)	CUERPO (2,n)
NOMFIL (n)	CUERPO (n,1)	CUERPO (n,2)	CUERPO (n,n)
	NOTACION (1)	NOTACION (2)	

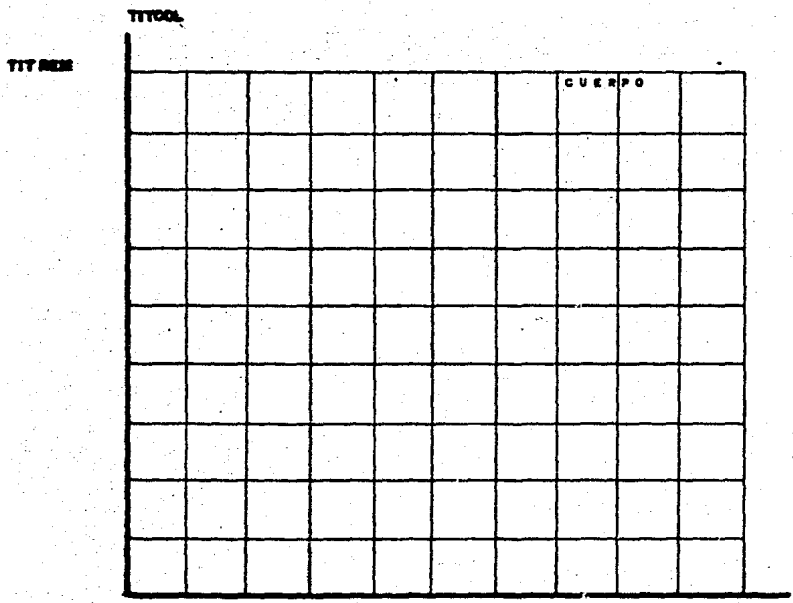
TITULO, TITCOL, TITREN, NOMCOL, NOMREN, MARCOCOL, MARCOREN y CUERPO se refieren a los campos indicados en la figura 3.1.

Los primeros tres campos se refieren a variables de tipo Texto; NOMCOL y NOMREN son arreglos de una dimensiòn de tipo Texto; MARCOREN y MARCOCOL son de tipo Arreglo Real Columna. CUERPO es de tipo Arreglo Real de dos dimensiones. Los límites inferiores de las dimensiones siempre serán uno y los superiores se estipulan en la declaraciòn.

En el caso del tipo Gràfica los títulos de las escalas pueden ser identificados mediante los campos TITCOL y TITREN. La Gràfica en sí la constituye el campo CUERPO. Obsèrvese la figura 3.2

FIGURA 3.2

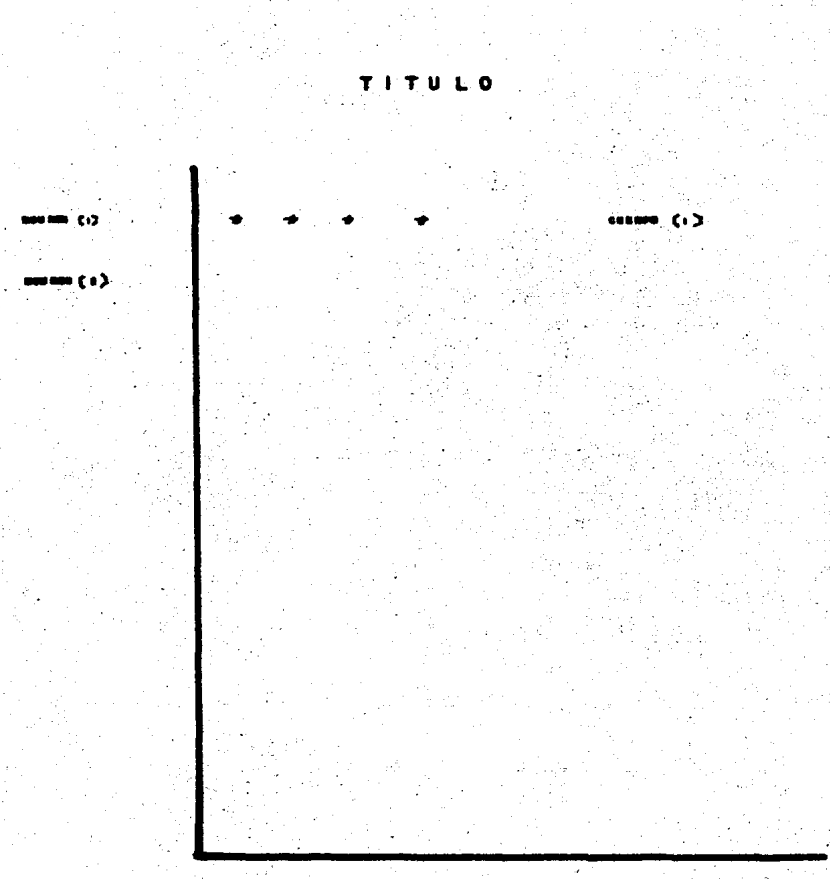
T I T U L O



TITULO, TITCOL, TITREN son de tipo Texto. El cuerpo de la gràfica es de tipo Cuerpo Gràfica. Este tipo conserva los puntos que se desean graficar en una matriz. En ella se encuentran marcados los puntos de la gràfica con el símbolo deseado almacenàndose en cuatro variables el màximo y el mínimo tanto de las ordenadas como de las abscisas con el objeto de reconstruir la escala. Vale la pena mencionar que se incluyò en el lenguaje una clase de expresiòn de tipo Cuerpo Gràfica la cual se compone de dos expresiones de tipo Arreglo Real de una dimensiòn que contienen las parejas de puntos a graficar y una expresiòn de tipo Texto con el símbolo con que se desea graficar.

En cuanto al tipo Histograma, las frecuencias (CUERPO) pueden ser identificadas mediante el campo NOMREN. Obsèrvese la figura 3.3

FIGURA 3.3



TITULO es de tipo Texto. NOMREN es un arreglo de una dimensiòn de tipo Texto mientras que CUERPO es un arreglo de una dimensiòn de tipo Real Columna. Los indices van de uno a los nùmeros estipulados en la declaraciòn. En la descripciòn de la proposiciòn de escritura se indica como se escriben estas estructuras.

3.2.5 Declaraciòn de Archivo y Formato

Por medio de estas declaraciones se especifican los archivos secuenciales que se utilizaràn, tanto de entrada como de salida y los formatos para las proposiciones de lectura y escritura.

```
<DECLARACION DE ARCHIVO>::=ARCHIVO
     <LISTA DE ARCHIVOS>
<LISTA DE ARCHIVOS>::=<IDENTIFICADOR>=
     <TEXTO>!<IDENTIFICADOR>=<TEXTO>,
      <LISTA DE ARCHIVOS>
<DECLARACION DE FORMATO>::=FORMATO
     <LISTA DE FORMATOS>
<LISTA DE FORMATOS>::=<DEFINICION DE FORMATO>!
     <DEFINICION DE FORMATO>,
     <LISTA DE FORMATOS>
<DEFINICION DE FORMATO>::=<IDENTIFICADOR>=
     <LISTA DE TEXTOS>
<LISTA DE TEXTOS>::=<TEXTO>!<TEXTO>,
     <LISTA DE TEXTOS>
```

La sintaxis de los caracteres de ediciòn, los cuales van en la <LISTA DE TEXTOS>, es la misma que la del lenguaje Fortran[D3], con la salvedad de que cuando se quieran utilizar comillas, estas se repetiràn dos veces. Por otro lado, el <TEXTO> en la <DECLARACION DE ARCHIVO> es el nombre externo del archivo al cual se asociarà el <IDENTIFICADOR> al principio del programa. El archivo se cerrarà al final del mismo.

Ejemplo:

```
ARCHIVO
   DATOS= "DATOSEDADES";

FORMATO
   CASO1="3X,5F5.2,2A6,/,4I5";
```

3.2.6 Declaraciòn de Rutina

Como ya se ha señalado, existen rutinas sin tipo y con tipo. Mientras que las primeras al ser llamadas se ejecuta el còdigo especificado en ellas, las segundas entregan ademàs un valor de tipo Real como resultado.

El còdigo especificado en las rutinas puede ser una proposiciòn compuesta o un bloque. En el caso de las rutinas con tipo debe existir por lo menos una asignaciòn que le de un valor al identificador de la rutina. Este valor serà el que entregue la rutina al ser llamada en una expresiòn.

En el encabezado de las rutinas se especifica si es con tipo o sin tipo, su nombre y un nùmero fijo de paràmetros, si los hay.

En las rutinas, uno se puede referir a algùn paràmetro mediante un identificador el cual constituye el paràmetro formal.

Existen dos clases de paràmetros: Llamados por valor y llamados por referencia. Ùnicamente los paràmetros de tipo Real podràn ser llamados por valor.

En el primer caso el paràmetro formal harà las veces de una variable local cuyo valor inicial es el resultado de una expresiòn que se evalùa antes de ejecutarse la rutina y que se encuentra en su lugar posicional en el llamado a dicha rutina. Esta expresiòn constituye el paràmetro actual.

En el segundo caso, el lugar de la expresiòn lo ocuparà un objeto y el paràmetro formal servirà para referir este objeto durante la ejecuciòn de toda la rutina.

```
<DECLARACION DE RUTINA>::=RUTINA <TIPO RUTINA>
     <IDENTIFICADOR><ZONA DE PARAMETROS
          FORMALES><CUERPO RUTINA>
<TIPO RUTINA>::= REAL ! <VACIO>
<ZONA DE PARAMETROS FORMALES>::=
        (<LISTA DE PARAMETROS FORMALES>) !
        <VACIO>
<LISTA DE PARAMETROS FORMALES>::=
        <TIPO PARAMETRO><LISTA DE
             IDENTIFICADORES> !
        <TIPO PARAMETRO><LISTA
             IDENTIFICADORES> <LISTA DE
               PARAMETROS FORMALES>
<TIPO PARAMETRO>::= VAL REAL ! <TIPO>
<TIPO>::=REAL ! TEXTO ! GRAFICA ! TABLA !
     HISTOGRAMA ! ARREGLO <TIPO ARREGLO>
               [ <NUMERO DIMENSIONES> ]
<NUMERO DIMENSIONES>::=,<NUMERO DIMENSIONES> !
     <VACIO>
<CUERPO RUTINA>::=<BLOQUE> ! <PROPOSICION COMPUESTA>
```

Anteponiendo la palabra reservada VAL al paràmetro se indica que es llamado por valor, por omisiòn el llamado es por referencia. Obsèrvese como se indica el tipo del paràmetro cuando este es arreglo, ùnicamente se especifica el nùmero de dimensiones, en forma similar a como se hace en Algol 68 [T1]

Ejemplo:

```
RUTINA REAL DENSIDAD(ARREGLO [] A,B
                     GRAFICA G1
                     REAL    R  )
  INICIO
     .
     .
     .
  FIN;
```

Para referirse a elementos de estructuras que sean paràmetros, se asumirà que el primer elemento de cada dimensiòn està en el indice uno.

3.3 Expresiones

Las expresiones se ubican en ciertos contextos, por ejemplo en la parte derecha del simbolo de asignaciòn, como paràmetros actuales en las llamadas a rutinas, etc. Para evaluar las expresiones es importante tomar en cuenta la precedencia de los operadores la cual es igual a la de la mayoria de los otros lenguajes de programaciòn. La fòrmula que define la expresiòn nos indica la precedencia de los operadores; no està definido el orden de evaluaciòn de secuencias de operadores con la misma precedencia. La razòn de esto es para no coartar la labor de una eventual optimizaciòn en la fase de implementaciòn. Analicemos primeràmente los operadores que actùan sobre los elementos de las dimensiones en las estructuras de tipo Arreglo. La sintaxis de estos operadores es la siguiente

```
<EXPRESION SELECTIVA DE ARREGLO>::=
     [<LISTA DE DIMENSIONES>]
<LISTA DE DIMENSIONES>::=
     <ELEMENTOS DE DIMENSION Y PUNTO> !
     <ELEMENTOS DE DIMENSION Y PUNTO>,
     <LISTA DE DIMENSIONES>
<ELEMENTOS DE DIMENSION Y PUNTO>::=
      . ! <ELEMENTOS DE DIMENSION> !
      <VACIO>
<ELEMENTOS DE DIMENSION>::=
     <GRUPO DE ELEMENTOS>!
     <GRUPO DE ELEMENTOS> &
```

<ELEMENTOS DE DIMENSION>
<GRUPO DE ELEMENTOS>::=<EXPRESION>!
<EXPRESION>_<EXPRESION>

La siguiente tabla nos define su funcionamiento:

OPERADOR	NOMBRE	DESCRIPCION
	PUNTO	Suma los elementos de la dimensiòn donde se encuentre. En el resultado esa dimensiòn quedarà reducida a un elemento. Sòlo se utilizarà en arreglos de tipo Real. Este operador se tomò de la forma en que se denota la operaciòn sobre todo en el campo de Diseño de Experimentos como puede verse en [H1], [F1] y [K1].
		Para seleccionar elementos de cada dimensiòn tenemos:
&	SELECTOR	Con el selector se pueden seleccionar uno a uno intervalos o elementos.
_	INTERVALO	La expresiòn del lado izquierdo del operador deberà ser menor o igual a la del lado derecho. Con este operador se selecciona un rango de elementos de una determinada dimensiòn. Este operador resuelve el problema de obtener submatrices.

El número de elementos seleccionados, no pudiéndose èstos repetir, serà el número de elementos de esa dimensiòn. El orden en que se seleccionen los elementos no alterarà su orden original. Si el lugar de la dimensiòn se deja vacio, se seleccionaràn todos los elementos de esa dimensiòn. Si existe una expresiòn esta funcionarà como un índice.

Ejemplo:

Sea A un arreglo declarado así:

```
ARREGLO
   A[1:5,2:8,0:10];
```

Entonces:

A[. ,2 & 4_6,]

quiere decir sumar los renglones, seleccionar las columnas 2, 4, 5 y 6 y tomar todos los planos.

Cabe preguntar aquí de què tipo serà el resultado de A[.,2&4_6,].

Al utilizar los operadores mencionados se efectùa una operaciòn, la cual denominaremos DIMENSIONAR con el objeto de obtener el tipo de una expresiòn o una estructura donde se utilicen estos operadores. Esta operaciòn consiste en lo siguiente:

Primeramente se cuentan los elementos de cada dimensiòn. Si una dimensiòn cuenta sòlo con un elemento, el nùmero de dimensiones quedarà reducido en uno.

Si el nùmero de dimensiones queda reducido a cero el tipo del resultado serà Real o Texto.

Si el nùmero de dimensiones es uno, en el caso de los arreglos reales, para definir si es renglòn o columna se sigue lo siguiente:

a) Si el arreglo està declarado como renglòn o columna, la declaraciòn nos lo define.

b) Sea p un nùmero entero mayor que uno, si el nùmero de elementos por dimensiòn resulta ser: quedarà definido como:

1,1,...,1,p	RENGLON
p,1,1,...,1	COLUMNA
1,1,...,1,p,1,1,...1	COLUMNA

Por lo tanto A[.,2&4_6,] es de tipo Arreglo Real de 2 dimensiones, mientras que por ejemplo A[.,,.] es de tipo Arreglo Real Columna y A[.,2,] es de tipo Arreglo Real Renglòn.

Esta operaciòn tambièn tiene lugar siempre que se utilicen arreglos o expresiones que en su declaraciòn o resultado respectivamente, tengan una o màs dimensiones con un solo elemento.

Con el objeto de ser consistentes en el uso de esta

operaciòn, cuando se utilice la <EXPRESION SELECTIVA DE ARREGLO> sobre un arreglo, èste primeramente se dimensiona. Esto se hace notar por los arreglos que se declaren con una o màs dimensiones de cardinalidad uno.

La <EXPRESION SELECTIVA DE ARREGLO> tambièn se puede utilizar para seleccionar (o sumar) elementos de una expresiòn cuyo resultado sea de tipo Arreglo, siendo esta expresiòn diferente a un arreglo declarado, como se verà màs adelante. En este caso, antes de aplicar la <EXPRESION SELECTIVA DE ARREGLO>, la expresiòn con valor de tipo Arreglo se dimensiona y se asume que el primer elemento de cada una de sus dimensiones ocupa el Indice uno.

En las expresiones se pueden incluir operandos constantes de tipo Arreglo n-dimensional ya sea de tipo Real o Texto

```
<ARREGLO EXPLICITO>::=[[<LIMITES EXPLICITOS>:
     <LISTA DE EXPRESIONES REPETITIVAS>]]
<LIMITES EXPLICITOS>::=<NUMERO ENTERO>!
     <NUMERO ENTERO>,<LIMITES EXPLICITOS>
<LISTA DE EXPRESIONES REPETITIVAS>::=
     <EXPRESION REPETITIVA> !
     <EXPRESION REPETITIVA>,
     <LISTA DE EXPRESIONES REPETITIVAS>
<EXPRESION REPETITIVA>::=<NUMERO ENTERO>
     (<LISTA DE EXPRESIONES REPETITIVAS>) !
      <EXPRESION>
```

El tipo de la <EXPRESION> deberà ser Real o (exclusivo) Texto para cada <ARREGLO EXPLICITO>.

La cantidad de nûmeros en <LIMITES EXPLICITOS> nos dice el nûmero de dimensiones del arreglo y los <NUMERO ENTERO>s, la cantidad de elementos por dimensiòn.

Si los <LIMITES EXPLICITOS> son n, p y q (nûmeros mayores que uno), el primer elemento de <LISTA DE EXPRESIONES REPETITIVAS> serà 111 el segundo elemento el 112 hasta el 11q el siguiente serà el 121, etc. Si faltasen elementos en la lista, los restantes seràn ceros y si sobrasen, èstos no se tomaràn en cuenta.

Si el arreglo es Real de una dimensiòn èste serà de tipo Arreglo Real Columna.

El <NUMERO ENTERO> en <EXPRESION REPETITIVA> nos indica que la <LISTA DE EXPRESIONES REPETITIVAS> se repite tantas veces como lo indica.

Ejemplo:

La matriz UNO de tres por tres sería:

[[3,3:3(1,0,0,0)]]

Veamos ahora la sintaxis de una expresión :

```
<EXPRESION>::=<EXPRESION SIMPLE> !
      <EXPRESION SIMPLE><OP. DE RELACION>
      <EXPRESION SIMPLE>
<OP. DE RELACION>::= <= ! < ! = ! <> !
      >= ! >
<EXPRESION SIMPLE>::= <OPERADOR UNARIO><TERMINO> !
      <OPERADOR UNARIO><TERMINO><FIN EXPRESION
        SIMPLE>
<FIN EXPRESION SIMPLE>::=<OPERADOR DE SUMA><TERMINO> !
      <OPERADOR DE SUMA><TERMINO><FIN
        EXPRESION SIMPLE>
<OPERADOR UNARIO>::= + ! - ! <VACIO>
<OPERADOR DE SUMA>::= + ! - ! O
<TERMINO>::=<FACTOR> ! <FACTOR><OPERADOR
       DE MULTIPLICACION><TERMINO>
<OPERADOR DE MULTIPLICACION>::= * ! / ! >< ! Y
<FACTOR>::=<OPERANDO> ! <OPERANDO> ** <OPERANDO> !
      <OPERANDO>@ ! <OPERANDO>'
<OPERANDO>::=<ELEMENTO PRIMARIO> !
      <ELEMENTO PRIMARIO><EXPRESION
        SELECTIVA DE ARREGLO> ! NO<OPERANDO>
<ELEMENTO PRIMARIO>::=<NUMERO> ! <TEXTO> !
      <IDENTIFICADOR DE CONSTANTE> !
      <CONSTANTE INTRINSECA> !
      <ARREGLO EXPLICITO> !
      (* DETERMINANTE : !<EXPRESION>! *) !
      (<EXPRESION>) !
      <EXPRESION TIPO CUERPO GRAFICA> !
      <IDENTIFICADOR DE RUTINA CON TIPO> !
      <IDENTIFICADOR DE RUTINA CON TIPO>
        <ZONA DE PARAMETROS> !
      <IDENTIFICADOR DE ARREGLO> !
      <IDENTIFICADOR DE ESTRUCTURA> !
      <IDENTIFICADOR DE ESTRUCTURA><CAMPO> !
      <IDENTIFICADOR DE VARIABLE> !
<CONSTANTE INTRINSECA>::= PI ! E
<CAMPO>::= .<CAMPO ESTRUCTURA>
<CAMPO ESTRUCTURA>::= TITULO ! TITCOL !
      TITREN ! NOMREN ! NOMCOL ! MARCOREN !
      MARCOCOL ! CUERPO
<ZONA DE PARAMETROS>::= (<PARAMETROS ACTUALES>)
<PARAMETROS ACTUALES>::= <EXPRESION> !
      <EXPRESION>,<PARAMETROS ACTUALES>
```

<EXPRESION TIPO CUERPO GRAFICA>::=
(<EXPRESION>,<EXPRESION>,<EXPRESION>)
<IDENTIFICADOR DE ARREGLO>::=<IDENTIFICADOR>
<IDENTIFICADOR DE ESTRUCTURA>::=<IDENTIFICADOR>
<IDENTIFICADOR DE VARIABLE>::=<IDENTIFICADOR>
<IDENTIFICADOR DE RUTINA CON TIPO>::=
<IDENTIFICADOR>

Veamos el significado de los operadores. La siguiente tabla nos define su funcionamiento. Cabe aclarar que cuando el orden de los operandos sea importante, se especificará en la descripción y que el tipo que aparece bajo TIPO DE SUS OPERANDOS es el que resulta después de dimensionar los de tipo Arreglo.

OPERADOR	NOMBRE	TIPO DE SUS OPERANDOS OP. 1	OP. 2
NO	NEGACION	REAL	
		ARREGLO REAL N-DIMENSIONAL	

Descripciòn: El resultado es del tipo del operando. Los nûmeros que sean iguales a cero en el operando, serân uno en el resultado, y los que sean distintos a cero, se tornarân cero. Este operador al igual que los dos siguientes, està orientado para ser usado en operaciones booleanas.

Y	Y	REAL	REAL
		ARREGLO REAL N-DIMENSIONAL	ARREGLO REAL N-DIMENSIONAL
		REAL	ARREGLO REAL N-DIMENSIONAL

Descripciòn: En los primeros 2 casos el resultado es del tipo de los operandos. En el segundo caso, cuando N sea igual a uno los dos operandos deberân ser renglòn o columna. Si se desea operar un Arreglo Real Renglòn con uno Columna, se aplicarà el operador "'" (traspuesta) a cualquiera de los dos operandos segûn como se desee el tipo del resultado. Las estructuras se operarân elemento a elemento siguiendo la tabla de verdad del operador lôgico Y siendo falso el valor cero. y verdadero los valores distintos a cero. En el resultado el valor de verdadero serà uno. Cuando en los operandos el nûmero de elementos en alguna dimensiòn sea distinto, el resultado conservarà el mayor y a los elementos sobrantes se les asignarà el valor cero. En el tercer caso el operando de tipo Real se aplicarà a cada uno de los elementos del operando de tipo Arreglo Real n-dimensional, siendo êste el tipo del resultado.

O	O	REAL	REAL
		ARREGLO REAL N-DIMENSIONAL	ARREGLO REAL N-DIMENSIONAL
		REAL	ARREGLO REAL N-DIMENSIONAL

Descripcion: Serà igual que el operador Y con la diferencia de que se usarà la tabla de verdad del operador O.

<=,<,=	OPERADORES DE RELACION	REAL	REAL
<>,>=,>		ARREGLO REAL	ARREGLO REAL
	(MENOR O IGUAL MENOR...)	N-DIMENSIONAL	N-DIMENSIONAL
		REAL	ARREGLO REAL N-DIMENSIONAL

Descripciòn: El orden de los operandos es importante. Se operarà elemento a elemento siguiendo la forma como se aplican los pasados operadores. Cuando la relaciòn sea cierta, en el resultado quedarà un uno, de lo contrario, cero. En el tercer caso el operando Real se relaciona con cada uno de los elementos del Arreglo.

+,-	MAS,MENOS	REAL	
	(UNARIOS)	ARREGLO REAL N-DIMENSIONAL	

Descripciòn: Basta decir que en el segundo caso el operador se aplicarà a cada uno de los elementos.

+,-,*,/	SUMA,RESTA	REAL	REAL
	MULTIPLICACION DIVISION	ARREGLO REAL N-DIMENSIONAL	ARREGLO REAL N-DIMENSIONAL

		REAL	ARREGLO REAL N-DIMENSIONAL

Descripciòn: En el caso de los operadores - y / el orden de los operandos es importante. La forma de operar es igual que en los pasados operadores binarios.

**	EXPONENCIACION	REAL	REAL
		ARREGLO REAL N-DIMENSIONAL	ARREGLO REAL N-DIMENSIONAL
		REAL	ARREGLO REAL N-DIMENSIONAL

Descripciòn: La forma de operar es igual que en los pasados operadores binarios. Es importante observar detenidamente el tercer caso y resaltar que el orden importa al aplicar el operador

><	PRODUCTO MATRICIAL	REAL	REAL
		ARREGLO REAL DE UNA O DOS DIMENSIONES	ARREGLO REAL DE UNA O DOS DIMENSIONES
		REAL	ARREGLO REAL DE UNA O DOS DIMENSIONES

Descripciòn: El orden de los operandos es importante. Para los arreglos de una dimensiòn se considerarà si son renglòn o columna. Se aplicaràn las reglas del producto matricial[L1] considerándose que los operandos sean conformables.

@	MATRIZ INVERSA GENERALIZADA	REAL	ARREGLO REAL DE UNA O DOS DIMENSIONES

Descripciòn: El operando harà las veces de una matriz y el resultado serà la matriz inversa generalizada (tal y como se

define y calcula en [G1]). El operador se escribe en posición sufija.

TRASPUESTA	REAL	ARREGLO REAL DE UNA O DOS DIMENSIONES	

Descripción: Las columnas y los renglones se intercambiarán. Si el operando es de tipo Real se deja igual. El operador se escribe en posición sufija.

!...!	DETERMINANTE	ARREGLO REAL DE DOS DIMENSIONES	

Descripción: El número de elementos de las dos dimensiones deberá ser igual. El operando se ubica entre los dos símbolos "!"

+	SUMA DE GRAFICAS	CUERPO GRAFICA	CUERPO GRAFICA

Descripción: Mediante este operador se pueden unir en un mismo plano dos o más gráficas (ver sección 3.2.4). Si dos o más gráficas distintas coinciden en un mismo punto, el símbolo que se pondrá en la posición del punto será cualquiera de los símbolos asociados a los puntos.

Veamos los siguientes ejemplos tomados de la bibliografía existente en los temarios de los cursos de estadística:

Sea EDADES un Arreglo Real Columna que contiene una muestra de edades

```
(((EDADES>5) Y (EDADES<90))*EDADES)[ . ]
```

nos da la suma de las edades entre 5 y 90. Si esto lo dividimos entre

```
((EDADES>5) Y (EDADES<90))[ . ]
```

que representa el número de casos de edades entre 5 y 90, obtenemos la media muestral.

Sea MEDIA y CASOS dos variables de tipo Real con la media muestral y el número de casos respectivamente del arreglo EDADES. Para obtener la varianza muestral de las edades tenemos

```
((EDADES - MEDIA)**2)[ . ] / CASOS
```

Sea OBSERVACIONES un Arreglo Real Columna que contiene los números de las caras de un dado que se obtuvieron después de arrojarlo N veces. La frecuencia observada del i-esimo número es

```
(OBSERVACIONES = I)[ . ]
```

donde I es igual a i.

Sea FRECUENCIAS un Arreglo Real Columna con las frecuencias observadas del ejemplo anterior. Dado que las frecuencias esperadas son N/6 el valor de chi-cuadrada es

```
(((FRECUENCIAS - N/6)**2)/(N/6))[ . ]
```

Sea PREFERENCIAS una Tabla con dos renglones y cuatro columnas donde se encuentra reflejada una población en relación con sus preferencias políticas. Los renglones corresponden a los sexos y las columnas a los partidos.

```
PREFERENCIAS.CUERPO[.,.]
```

es el total de la población

```
(PREFERENCIAS.CUERPO[.,]/PREFERENCIAS.CUERPO[.,.])*100
```

representa un vector con los porcentajes de las preferencias por partido.

Sean X1 y Y1 dos arreglos de tipo Arreglo Real Columna con n elementos que representan dos vectores. El producto interior de X1 por Y1 sería

```
(X1 * Y1)[ . ]  o  X1' >< Y1
```

Sean X1, FX1 y X2, FX2 cuatro arreglos de tipo Real Columna con los puntos de las gráficas de dos funciones y sea G1 una estructura de tipo Gráfica

```
(X1,FX1,"*") + (X2,FX2,"@") + G1.CUERPO
```

representa la unión de las tres gráficas.

3.4 Proposiciones

3.4.1 Asignación

La sintaxis de la proposición de asignación es la siguiente:

```
<PROPOSICION DE ASIGNACION>::=
      <ESTRUCTURA REFERENCIABLE> :=
          <EXPRESION>
<ESTRUCTURA REFERENCIABLE>::=<IDENTIFICADOR
      DE RUTINA CON TIPO> !
      <IDENTIFICADOR DE ARREGLO> !
      <IDENTIFICADOR DE ARREGLO><EXPRESION
       SELECTIVA DE ARREGLO> !
      <IDENTIFICADOR DE ESTRUCTURA> !
      <IDENTIFICADOR DE ESTRUCTURA><CAMPO> !
      <IDENTIFICADOR DE VARIABLE>
```

En la <ESTRUCTURA REFERENCIABLE> no podrá aparecer el operador PUNTO en la <EXPRESION SELECTIVA DE ARREGLO>.

También cuando se trate de estructuras tipo Arreglo, la estructura se dimensionará y la asignación se hará elemento a elemento. Si el número de elementos por dimensión no coincide se hará lo siguiente: Si la <EXPRESION> tiene menos elementos que la <ESTRUCTURA REFERENCIABLE> en alguna dimensión, los elementos que sobren de esta última se dejarán intactos, de lo contrario, los elementos que sobren en la <EXPRESION> no se tomarán en cuenta.

Ejemplo

Sea PREFERENCIAS la tabla del ejemplo en 3.3. Las asignaciones siguientes :

```
PREFERENCIAS.TITULO:="PREFERENCIAS"
 PREFERENCIAS.TITCOL:="PARTIDOS"
 PREFERENCIAS.TITREN:="SEXO"
 PREFERENCIAS.NOMREN:=[[2:"MASC","FEM"]]
 PREFERENCIAS.NOMCOL:=[[ 4 :
"PARTIDO 1","PARTIDO 2","PARTIDO 3",
"PARTIDO 4"]]
 PREFERENCIAS.MARCOREN:=[[2:1,1]]
 PREFERENCIAS.MARCOCOL:=[[4:4(1)]]
```

ponen los títulos de la tabla y enmarcan la información.

3.4.2 Proposición de Escritura

Esta proposición tiene como finalidad escribir en un archivo secuencial una lista de expresiones cuyo resultado sea de tipo Real, Texto, Arreglo Real y Texto de una dimensión, Gráfica, Tabla e Histograma.

```
<PROPOSICION DE ESCRITURA>::=
     ESCRIBE EN <IDENTIFICADOR> CON
     <FORMATO><ELEMENTOS A ESCRIBIR>
<FORMATO>::=LIBRE ! <LISTA DE TEXTOS> !
     <IDENTIFICADOR>
<ELEMENTOS A ESCRIBIR>::= ESTO
     <LISTA DE EXPRESIONES A ESCRIBIR> !
     <VACIO>
<LISTA DE EXPRESIONES A ESCRIBIR>::=
     <EXPRESION> ! <EXPRESION>,<LISTA DE
      EXPRESIONES A ESCRIBIR>
```

El <IDENTIFICADOR> que se encuentra después de EN debe ser un archivo.

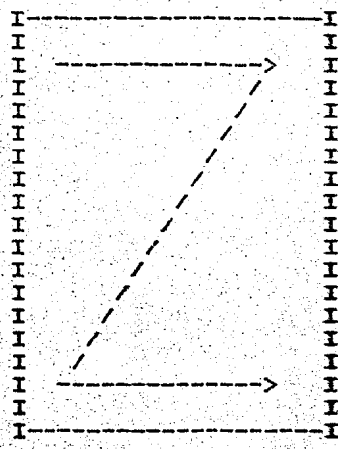
El <IDENTIFICADOR> que aparece en <FORMATO> debe ser un formato declarado. La <LISTA DE TEXTOS> debe contener caracteres de edición siguiendo la sintaxis de los formatos en Fortran[D3].

El formato LIBRE se utilizará al escribir una expresión de tipo Gráfica, Tabla o Histograma. La proposición distribuirá cada uno de los campos de la estructura como se encuentran en las figuras 3.1, 3.2 y 3.3. MARCOCOL y MARCOREN se escribirán si en la posición correspondiente existe un número distinto de cero.

Las escalas de las gráficas se calcularán utilizando el máximo y el mínimo tanto de las abscisas como de las ordenadas y éstas se escribirán en los ejes de la gráfica. En el caso del Histograma, las frecuencias se escribirán en el eje horizontal.

Si el tamaño del registro del archivo de salida no es lo suficientemente grande como para que pueda escribirse la estructura en él, se seguirá la dirección de impresión que se muestra en la figura 3.4

FIGURA 3.4



Ejemplo

```
ESCRIBE EN LISTADO CON LIBRE ESTO GRAFICA1

ESCRIBE EN DATOS CON "6(5X,",
  "F8.3" ESTO INFO[I,]
```

3.4.3 Proposición de Lectura

La proposición de asignación y la de lectura tienen la capacidad de modificar los valores de las distintas estructuras durante la ejecución de un programa.

```
<PROPOSICION DE LECTURA>::= LEE DE
     <IDENTIFICADOR><FIN ARCHIVO> CON
     <FORMATO><RECEPTORES>
<FIN ARCHIVO>::= FIN <IDENTIFICADOR> !
     <VACIO>
<RECEPTORES>::=ESTO <LISTA DE RECEPTORES> !
     <VACIO>
<LISTA DE RECEPTORES>::= <RECEPTOR> !
     <RECEPTOR>,<LISTA DE RECEPTORES>
<RECEPTOR>::=<ESTRUCTURA REFERENCIABLE>
```

El tipo de <RECEPTOR> debe ser Real, Texto, Arreglo Real y Texto de una dimensión.

<FIN ARCHIVO> se utilizará para indicar el fin de archivo. <IDENTIFICADOR> debe ser una variable de tipo Real cuyo valor será uno cuando suceda la condición de fin de archivo, si esta condición no sucede, esta variable conservará su valor.

Ejemplo

```
LEE DE DATOS CON CASO ESTO A[I,]

LEE DE DATOS FIN EOF CON "4F6.2"
     ESTO  X,A[I,2_4]
```

3.4.4 Proposición de Llamado a Rutina

Con esta proposición se ejecuta una rutina sin tipo. Esta rutina puede tener parámetros los cuales tendrán que especificarse en el llamado.

```
<PROPOSICION DE LLAMADO A RUTINA>::=
     <IDENTIFICADOR DE RUTINA SIN TIPO> !
     <IDENTIFICADOR DE RUTINA SIN TIPO>
     <ZONA DE PARAMETROS>
```

```
<IDENTIFICADOR DE RUTINA SIN TIPO>::=
      <IDENTIFICADOR>
```

Debido a que el cuerpo de una Tabla es de tipo Arreglo esta podrà pasarse como un parámetro de tipo Arreglo de 2 dimensiones, lo mismo sucede con los titulos de las estructuras, que pueden pasarse como textos o los nombres de columnas y renglones, como arreglos de tipo Texto.

Por otro lado, la <EXPRESION SELECTIVA DE ARREGLO> no podrà usarse en un parámetro actual excepto cuando se refiera a un solo elemento.

3.4.5 Proposiciòn Condicional

Como se ha mencionado, esta proposiciòn funciona en forma parecida al IF..THEN e IF..THEN..ELSE de Algol

```
<PROPOSICION CONDICIONAL>::=
      SI <EXPRESION> ENTONCES
       <PROPOSICION INCONDICIONAL> !
      SI <EXPRESION> ENTONCES
       <PROPOSICION INCONDICIONAL> SINO
       <PROPOSICION SIN ETIQUETA>
```

Si al evaluar la <EXPRESION>, cuyo resultado debe ser de tipo Real, resulta ser igual a cero, se efectuarà la alternativa. De no ser asi, se efectuarà la <PROPOSICION INCONDICIONAL> Nòtese que para evitar confusiones al hacer corresponder los SIs con sus respectivos SINOs, a continuaciòn de la palabra ENTONCES se encuentra una <PROPOSICION INCONDICIONAL>, razòn por la cual es invàlida una proposiciòn de la forma SI <EXPRESION> ENTONCES SI ...

Ejemplo

```
   SI !A!<>0 ENTONCES
     INICIO
       X:=A@ >< C;
       ESCRIBE EN SOLUCIONES CON FORM
           ESTO X
     FIN
```

3.4.6 Proposicion Selectiva

El funcionamiento de esta proposiciòn es similar al CASE de Pascal.

```
<PROPOSICION SELECTIVA>::=LA <EXPRESION>
```

```
      DE <PROPOSICION COMPUESTA ETIQUETADA> !
      LA <EXPRESION> DE <PROPOSICION
         COMPUESTA ETIQUETADA> SINO
        <PROPOSICION SIN ETIQUETA>
<PROPOSICION COMPUESTA ETIQUETADA>::=
      INICIO <LISTA DE PROPOSICIONES
         ETIQUETADAS> FIN
<LISTA DE PROPOSICIONES ETIQUETADAS>::=
      <LISTA DE ETIQUETAS>:<PROPOSICION SIN
       ETIQUETA> !
      <LISTA DE ETIQUETAS>:<PROPOSICION SIN
       ETIQUETA>;<LISTA DE PROPOSICIONES
       ETIQUETADAS>
<LISTA DE ETIQUETAS>::=<ETIQUETA> !
      <ETIQUETA>,<LISTA DE ETIQUETAS>
```

El resultado de la <EXPRESION> debe ser de tipo Real.

Según el valor redondeado de la <EXPRESION> el flujo de control se pasará a la <PROPOSICION SIN ETIQUETA> que se le haya antepuesto una <ETIQUETA> con ese valor. El flujo de control pasará a la alternativa si ninguna etiqueta coincide con el valor de la <EXPRESION>.

Ejemplo

```
      LA M DE
       INICIO
        1:TRASPUESTA:=X';
        2:INVERSA:=X@;
        3:DETERMINANTE:= ! X !
       FIN
      SINO
       ESCRIBE EN ARCHIVO CON
        "18HERROR EN OPCION : ,I5" ESTO M
```

3.4.7 Proposición Iterativa

La proposicion iterativa tiene varias opciones a saber:

```
<PROPOSICION ITERATIVA>::= REPITE
      <CUERPO ITERACION>
<CUERPO ITERACION>::=
      <PROPOSICION INCONDICIONAL>
         HASTA <EXPRESION> !
      CON <IDENTIFICADOR> := <EXPRESION>
         SUMANDO <EXPRESION> HASTA
         <EXPRESION> ESTO
         <PROPOSICION INCONDICIONAL> !
```

```
    MIENTRAS <EXPRESION> ESTO
       <PROPOSICION INCONDICIONAL>
```

Las alternativas de <CUERPO ITERACION> corresponden a las proposiciones DO..UNTIL, FOR..STEP..DO y WHILE..DO de Algol. La <EXPRESION> que precede a la palabra HASTA en la primera alternativa, debe ser de tipo Real. La <PROPOSICION INCONDICIONAL> se harà por lo menos una vez y si el valor de la <EXPRESION> es cero, se repetirà. En la segunda alternativa las expresiones y la variable de control son de tipo Real. En la alternativa MIENTRAS, la <PROPOSICION INCONDICIONAL> se repetirà mientras la <EXPRESION>, que debe ser de tipo Real, no sea cero.

Ejemplo

```
LEE DE DATOS FIN FINARCHIVO CON F ESTO A;
REPITE MIENTRAS NO FINARCHIVO ESTO
  INICIO
    ESCRIBE EN ARCHIVO CON F ESTO A;
    LEE DE DATOS FIN FINARCHIVO CON F ESTO A
  FIN;
```

3.4.8 Proposiciòn ver

Esta proposiciòn sirve para indicar que el control deberà pasar a otra parte del programa. Las etiquetas son numèricas, no se declaran y son estrictamente locales al bloque donde se encuentran.

<PROPOSICION VER>::= VER <ETIQUETA>

Ejemplo

```
    VER 5
```

3.4.9 Proposiciòn vacía

Esta proposiciòn denota la ausencia total de acciòn

<PROPOSICION VACIA>::=<VACIO>

Ejemplo

```
    INICIO
      A:=0;
    FIN
```

En esta <PROPOSICION COMPUESTA> tenemos que la <PROPOSICION VACIA> está entre el primer punto y coma y la <PALABRA RESERVADA> FIN

4.EL COMPILADOR

4.1 Introducción

Varios aspectos influyeron en la elaboración del compilador.

La computadora con la que se trabajó fue un factor decisivo. El lenguaje se quiso implementar en el laboratorio de estadística de la Facultad de Ciencias. Como ya se mencionó, el laboratorio cuenta con una minicomputadora NOVA 3/12 cuyas características se describen en el apéndice A. El utilizar esta computadora requirió resolver problemas como el de administrar correctamente la memoria, el de utilizar un lenguaje como es Fortran para escribir el compilador y los problemas propios del aprendizaje y manejo de los sistemas de la computadora.

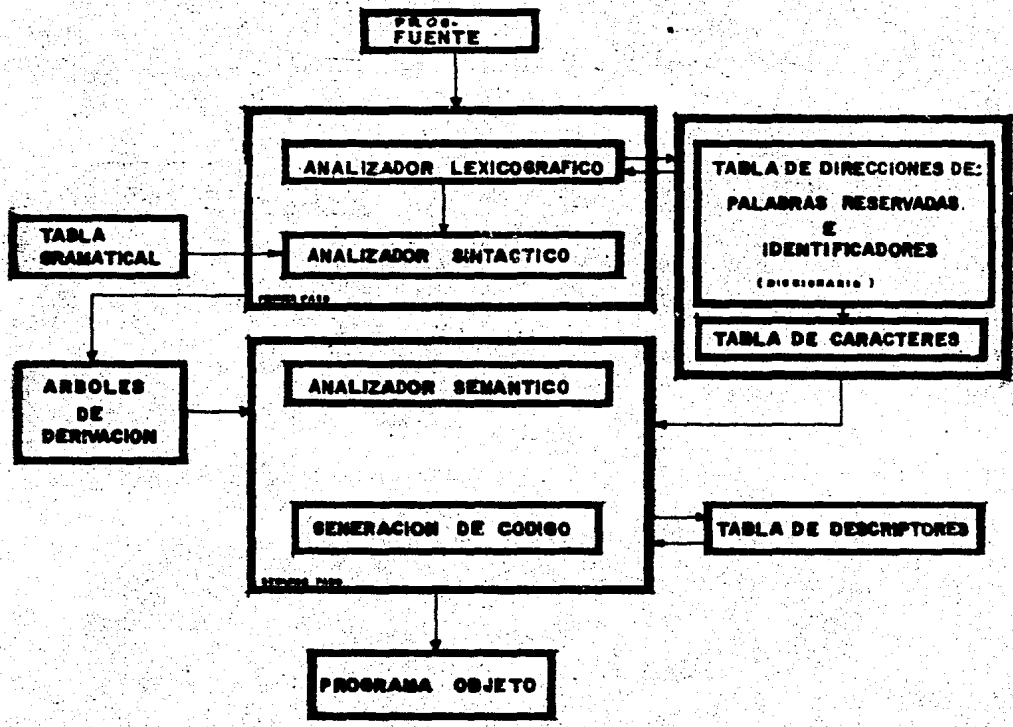
Una ventaja de haber utilizado el lenguaje Fortran para el desarrollo del compilador es que lo hace transportable. Las pruebas iniciales del análisis lexicográfico y sintáctico se realizaron en una máquina CDC, posteriormente se transportaron a la NOVA 3/12 citada anteriormente.

Se dió prioridad, sobre otros aspectos, a la flexibilidad del compilador con el objeto de que fuese fácil incorporar cambios durante el desarrollo, así como también el poder incorporar extensiones futuras.

El compilador se guía de una tabla gramatical para reconocer las distintas estructuras del programa fuente. Dicha tabla se genera mediante un programa a partir de una gramática escrita en un BNF modificado para facilitar la mecanografía. Una vez generada la tabla gramatical, el compilador puede ser utilizado.

La capacidad de la computadora determinó que el compilador se dividiera en dos pasos. El primer paso comprende el análisis lexicográfico y sintáctico generándose los árboles de derivación. El segundo paso abarca el análisis semántico y la generación de código. Inicialmente el compilador se hizo de un paso pero al ir desarrollándose tuvo que ser dividido debido a restricciones de memoria. El siguiente diagrama describe su estructura

FIGURA 4.1


PROG. FUENTE
ANALIZADOR LEXICOGRAFICO
TABLA DE DIRECCIONES DE:
PALABRAS RESERVADAS
E
IDENTIFICADORES
(DICCIONARIO)
TABLA GRAMATICAL
ANALIZADOR SINTACTICO
TABLA DE CARACTERES
ARBOLES DE DERIVACION
ANALIZADOR SEMANTICO
GENERACION DE CODIGO
TABLA DE DESCRIPTORES
PROGRAMA OBJETO

4.2 Analizador Lexicogràfico

La función principal del analizador lexicogràfico es reconocer los àtomos (tokens) que conforman el programa fuente y convertir éstos a una notación interna màs manipulable y compacta la cual llamaremos "texto limpio". Cada oración del programa fuente, llàmese declaración o proposición es tratada por el analizador. Una vez transformada la oración, el analizador sintàctico la toma y desarrolla con ella su función.

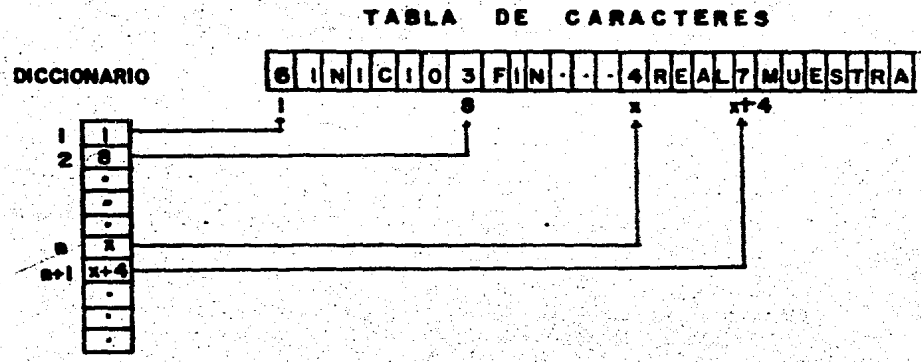
Al efectuar la traducción, el analizador lexicogràfico elimina los comentarios y los espacios superfluos e identifica los separadores de declaraciones y proposiciones.

Se vale del primer caracter de una cadena para determinar el tipo de àtomo de que se trate. El proceso es transferido a la rutina correspondiente que termina con el recorrido izquierda - derecha de la cadena complementàndose así la total identificación del àtomo. Los reconocedores de cada uno de los àtomos corresponden a un autòmata reconocedor de gramàticas regulares [D5].

Si el primer caracter es una letra, se asume que el àtomo a reconocer será una palabra reservada o un identificador. Debido a la sencillez de la sintaxis de estos àtomos el analizador lo reconoce en forma ad hoc facilitando de esta forma el trabajo del analizador sintàctico.

Posteriormente, estos àtomos se traducen a una pareja de números formada por un còdigo seguido de su posición en una tabla que llamaremos "diccionario" a la cual son incorporados. Esta contiene referencias a una tabla de caracteres donde se encuentran codificadas las palabras reservadas y los identificadores tal y como aparecen en el programa fuente. Al inicio del anàlisis el "diccionario" se inicializa con todas las palabras reservadas. Después de inicializado el "diccionario", se conserva la última posición ocupada la cual servirà para establecer la diferencia entre palabras reservadas e identificadores. Si la posición del elemento en el "diccionario" es mayor a la última posición ocupada por una palabra reservada, implica que la cadena es un identificador. Observar la figura 4.2

FIGURA 4.2



n= Posición de la última palabra reservada
n+1= Posición del primer identificador

Antes de incorporar un nuevo elemento al "diccionario", el analizador verifica que anteriormente no haya sido incluido valiéndose de la tabla de caracteres. Si èste ya hubiese sido incorporado, ùnicamente entrega al "texto limpio" su posiciòn en el "diccionario". La posiciòn de las palabras reservadas e identificadores determina plenamente de cual se trata para efectos de su reconocimiento por parte de los analizadores sintàctico y semàntico.

Por otro lado, las comillas (") marcan el inicio de un texto. Este se transformarà a un còdigo y su posiciòn en la tabla de caracteres a la cual se incorpora.

Un digito serà el inicio de un nùmero. Los nùmeros se traduciràn a un còdigo y un apuntador a la tabla de caracteres donde se incorporan.

El caracter "%" marcarà el inicio de un comentario y se ignoraràn todos los caracteres hasta encontrar un ";". Los blancos tambièn son ignorados.

Todos los demàs caracteres se traducen tal y como aparecen en el programa fuente a excepciòn del ";" que separa declaraciones y proposiciones. Al identificarse este simbolo el analizador sintàctico empezarà su labor reconociendo el "texto limpio".

Veamos el siguiente ejemplo en la figura 4.3

FIGURA 4.3

Declaración de cuatro variables de tipo REAL

```
(  2)  REAL
(  3)          X,YY,I,FINARCHIVO;
   2000    4   1020    52    44   1030    53    44   1020
```

El código 2000 se utiliza para indicar que el átomo es una palabra reservada. Los distintos códigos que indican que clase de átomo viene a continuación son los siguientes:

2000	Palabra Reservada
1020	Identificador
1010	Número
1040	Texto

El número que se encuentra a continuación del código 2000 identifica de que palabra reservada se trata. En el apéndice C sección C.4 se encuentra una lista de todas las palabras reservadas. Los códigos para cada una de éstas estan en la columna LOC.IVAR. Como puede observarse, el número 4 corresponde a la palabra reservada REAL

Identificador X. El siguiente número, 52, es la posición del identificador en el Diccionario.

Caracter ",". Todos los caracteres se representan utilizando su respectivo código ASCII.

Caracter ";"

```
54    44    1020    55    59
```

4.3 Analizador Sintàctico

La función primordial del analizador sintáctico es elaborar el árbol de derivación o sintáctico de cada una de las oraciones del programa fuente. Asi mismo verifica que todas las construcciones sean válidas. Para esto necesita contar con el "texto limpio" elaborado por el analizador lexicográfico.

El analizador sintáctico se guia de una tabla gramatical para reconocer el "texto limpio". Con el objeto de que la incorporación de cambios a la gramática durante su desarrollo fuese sencilla, el compilador se diseñó para que al inicio de su ejecución leyera la tabla gramatical de un archivo.

Para la construcción de dicha tabla se elaboró un programa cuya función es tomar una gramática escrita en un BNF levemente modificado y generar la tabla en una representación interna ya utilizable por el compilador.

Asi, al introducir cambios a la gramática, basta con modificar la gramática escrita en nuestro BNF.

Antes de presentar el funcionamiento del analizador veamos la gramática que maneja, sus características y su representación interna.

4.3.1 Gramática

Para facilitar el reconocimiento de un programa, el analizador sintáctico se diseñó para que trabajase sobre una declaración o sobre una proposición a la vez. Debido a esto, la gramática del lenguaje fue modificada para que el símbolo distinguido fuese un símbolo no terminal que comprendiese la definición de una declaración y una proposición. La gramática se encuentra descrita en el apéndice C.

Para describirla se utilizó un metalenguaje similar a BNF. La siguiente tabla nos da la equivalencia de símbolos:

Símbolo	Símbolo BNF
(	<
)	>

!	! (alternativa)
=	::=

Asi mismo los simbolos terminales se rodearon de apòstrofes. Se utilizò un ";" para separar las producciones y el simbolo "*" para dar por terminada la gramàtica. Otra diferencia es la siguiente: Como se puede observar, a las producciones les antecede una B o una P. Las producciones del tipo B() asocian un nùmero predefinido a un simbolo no terminal. Estos nùmeros estàn relacionados con la traducciòn que hace el analizador lexicogràfico de ciertos àtomos que èl mismo reconoce como son los identificadores, las palabras reservadas, los nùmeros y los textos. Por ejemplo un identificador, como ya vimos, se traduce a la pareja (1020,x) donde 1020 es el còdigo que se utiliza para indicar que se trata de un identificador y la x es su posiciòn en el "diccionario". Esta pareja se inserta en el "texto limpio". Si el identificador se ubica bien sintàcticamente en el programa fuente, deberà ser reconocida por (IDENT) el cuàl se define como B(IDENT)=1020. Este tipo de producciones sirve, por lo tanto, para reconocer àtomos predefinidos[R1].

Las producciones del tipo P() son equivalentes a aquellas escritas en BNF.

Para que esta gramàtica pudiese ser utilizada por el analizador sintàctico fue necesario traducirla a una representaciòn interna que constituirà la tabla gramatical. Esta representaciòn consiste en lo siguiente:

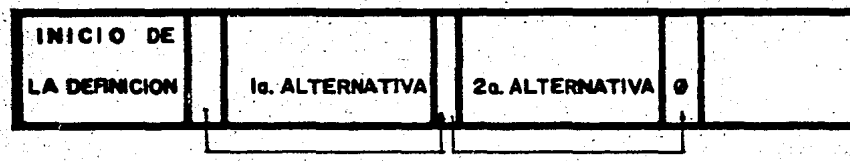
Un simbolo no terminal quedarà definido una sola vez siendo su valor interno la posiciòn que le asigna el programa traductor de la gramàtica dentro de la tabla y dicho valor a su vez es un apuntador a la definiciòn del simbolo.

Los simbolos terminales adoptan su respectivo còdigo ASCII.

Los simbolos definidos por una producciòn del tipo B() se consideraràn como terminales y en su lugar aparecerà el nùmero asociado.

Una producciòn del tipo P() serà representada como se muestra en la figura 4.4 [L2]

FIGURA 4.4

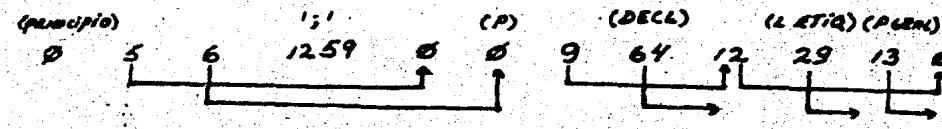


Ejemplo:

P(PRINCIPIO) = (P) ';' ;

P(P) = (DECL) ! (L ETIQ)(P GRAL) ;

SE TRADUJO A:



4.3.2 Anàlisis Sintàctico

La tabla gramatical es incorporada al analizador al inicio del proceso. Así mismo se inicializa el "diccionario" con las palabras reservadas.

En base a esta gramàtica y haciendo un anàlisis desde arriba (top-down) se crean los àrboles de derivaciòn de las diferentes oraciones representadas en el "texto limpio". Para esto el analizador navega por la gramàtica localizando los símbolos terminales y comparàndolos contra los que se tienen. Por la forma en que lo hace es necesario eliminar el uso de producciones con recursiòn izquierda, es por eso que como puede observarse, la gramàtica es recursiva derecha.

Si el analizador logra reconocer el "texto limpio" mediante una tècnica de "prueba y error" (backtrack), crea una representaciòn lineal de las alternativas de la gramàtica que se recorrieron para reconocer la oraciòn.

Este recorrido representa el àrbol de derivaciòn o sintàctico y es el que toma el analizador semàntico.

Para auxiliar al analizador semàntico en el recorrido del àrbol, se le pasa tambièn un vector de apuntadores que señala, para cada nodo del àrbol, donde se ubica su hermano.

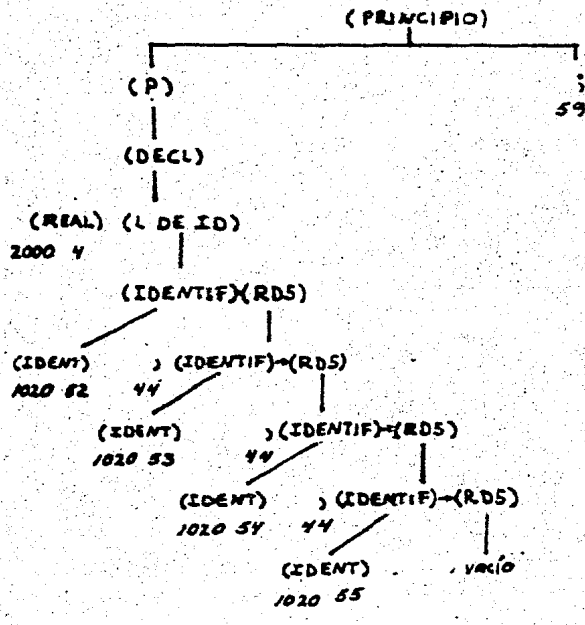
En el siguiente ejemplo (figura 4.5) tenemos la declaraciòn de cuatro variables de tipo Real y el texto limpio generado por el analizador lexicogràfico. A continuaciòn aparece el àrbol de derivaciòn como lo construye el compilador (1a. linea: un nùmero secuencial de referencia, 2a. linea: el àrbol de derivaciòn y 3a. linea: el vector de apuntadores a los hermanos) y la gràfica del mismo àrbol.

FIGURA 4.5

```
(  2)  REAL
(  2)          X,YY,I,FINARCHIVO;
      2000        4    1020        52        44    1020        53        44    1020
        54       44    1020        55        59
1a.    1         2       3         4         5         6         7         8         9        10
2a.    1         1       2         1        52         1        53         1        54         1
3a.   13        13      13        13         6        13         8        13        10        13

      11        12
      55         2
      12        13
```

Gráfica del árbol de derivación de acuerdo con la gramática definida en el apéndice C sección C.1



4.4 Analizador Semàntico y Generaciòn de Còdigo.

En esta parte del compilador se obtiene el "significado" de los programas, se revisan si estàn bien semànticamente hablando detectàndose errores que se derivan de la inherente dependencia de contexto del lenguaje como identificadores no declarados o ya declarados, concordancia de tipos en expresiones, etc. a la vez de que se elaboran una serie de estructuras como tablas de descriptores, , etiquetas, etc. que serviràn para la sintesis del programa objeto.

Asi como el analizador sintàctico se guia de la tabla gramatical, el analizador semàntico se guia de los àrboles de derivaciòn para realizar su funciòn.

Es importante hacer notar que la estructura del analizador sigue la estructura de la gramàtica y es dirigido por el àrbol sintàctico de manera que al modificar la gramàtica resulta relativamente fàcil modificar el analizador.

Parte fundamental del analizador semàntico es la tabla de descriptores o simbolos. Por medio de esta tabla el analizador almacena toda la informaciòn necesaria acerca de las estructuras de datos que se utilizaràn en el programa fuente para elaborar sus equivalentes en el programa objeto; se basa tambièn en ella para verificar la semàntica del programa. Al respecto Gries dice lo siguiente [G2]

> *"Todo compilador usa una tabla de simbolos (algunas veces llamada lista de identificadores o tabla de nombres) en una forma u otra. Es una tabla de identificadores usados en el programa fuente junto con sus atributos. Los atributos son el tipo del identificador, su direcciòn en el programa objeto y cualquier otra informaciòn acerca de èl necesaria para generar còdigo"*

Debido a la importancia de esta estructura se decidiò implementar esta parte del analizador semàntico. Primeramente veamos la organizaciòn de la tabla de descriptores.

4.4.1 Organizaciòn de la Tabla de Descriptores.

Dada la estructura de bloques del lenguaje, la tabla de descriptores se organiza de la siguiente manera:

Al inicio de cada bloque la tabla, que funciona como una pila, es marcada indicando un nuevo nivel en el cuàl un conjunto de identificadores con sus respectivos atributos serán incorporados.

Al encontrarse con una declaración, el analizador toma los atributos del identificador y los almacena en la tabla en el nivel que está en el tope, verificando que el nuevo elemento no sea igual a ninguno de los de ese nivel pues de ocurrir ésto habría un error de identificador doblemente declarado. Para facilitar la búsqueda en la tabla de descriptores se utilizan una serie de estructuras que a continuación se explican.

Dado que el analizador no recibe del árbol sintáctico el identificador en sí, sino un apuntador al "diccionario", son éstos los que se asocian con los respectivos atributos de los identificadores. Cada apuntador, que corresponde a un identificador, se liga con una lista de apuntadores que referencian la tabla de descriptores. Cada elemento de la lista representa una estructura diferente, bajo un mismo identificador en distintos bloques anidados. (Es obvio que en la lista no hay dos elementos que apuntan a descriptores de un mismo nivel).

En cualquier momento, el primer elemento de la lista apunta al descriptor que se debe utilizar. Gracias a esta organización, para encontrar el descriptor de un identificador basta consultar el primer elemento de su lista, así mismo, es fácil saber si está doblemente declarado (si al tratar de incorporarlo a la tabla el apuntador del primer elemento de la lista es mayor que la primera localidad del nivel donde se está ubicando) o si no está declarado (si la lista asociada al identificador es vacía). De esta forma también se resuelve la referencia a estructuras no locales a un determinado bloque.

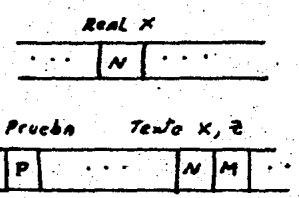
Al terminar el bloque, el nivel que está en el tope de la tabla se libera y se actualizan las listas de los identificadores de ese nivel utilizando un apuntador al "diccionario" que se encuentra al principio de cada descriptor. Observar el ejemplo de la figura 4.6

FIGURA 4.6

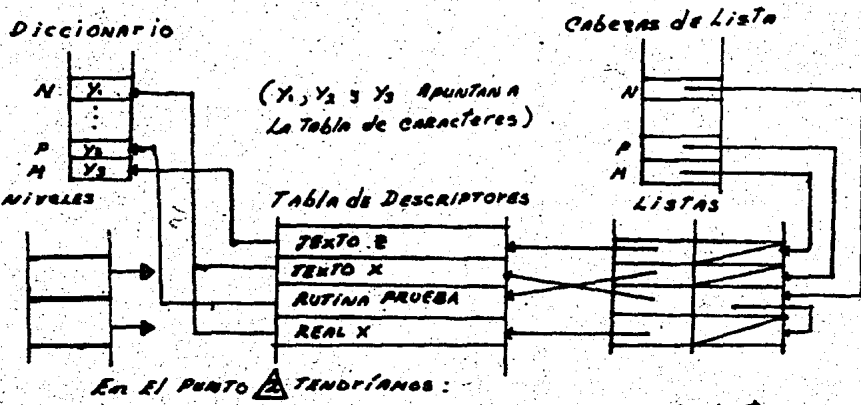
Supóngase que se tiene lo siguiente:

```
inicio
  REAL x;
  :
  RUTINA prueba;
  inicio
    TEXTO x, z;
  ①
  FIN
②
```

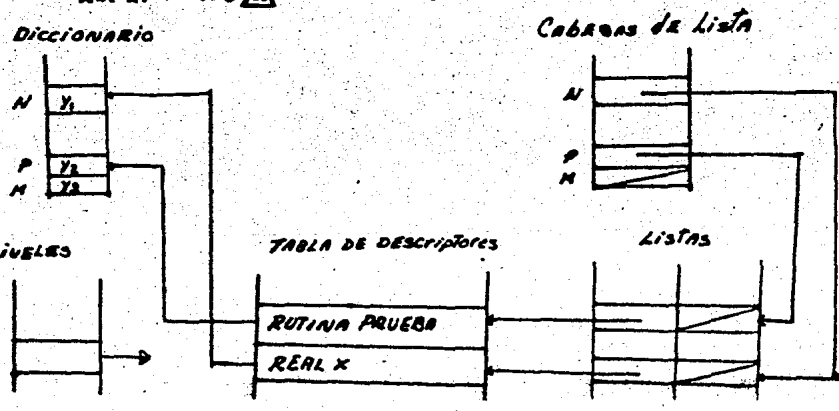
Árboles de derivación:



EN EL PUNTO ① TENDRÍAMOS:



En el punto ② tendríamos:



A continuación se presentan los descriptores de cada una de las estructuras del lenguaje.

4.4.2 Descriptores.

En este punto quisiéramos resaltar que el corazón del analizador semántico es la tabla de descriptores pues es la base para la generación de código.

Cada descriptor se compone de dos partes:

La primera se compone de:

1) Tipo
2) Forma
3) Localidad en memoria o referencia al nombre interno de una rutina o archivo.
4) Número de campos, dimensiones o parámetros.

El tipo, como su nombre lo indica, señala cuál es el tipo de la estructura de datos que está describiendo (Real, Texto, Tabla, Gráfica, Histograma, Archivo, Formato, Rutina con tipo o sin tipo).

La forma nos da información acerca de si la estructura es una constante, arreglo, variable simple, si es un parámetro formal llamado por valor o referencia, etc.

El tercer campo nos indica la localidad que la estructura ocupará en el programa objeto. Se asume que todas las estructuras del programa fuente estarán en el programa objeto en un arreglo lineal que funcionará como una pila al cual llamaremos Memoria. En el caso de las rutinas, las cuales se traducirán a rutinas en Fortran, este campo nos indica el nombre de la subrutina correspondiente. Para los archivos, este campo será su nombre lógico en el programa objeto. En el caso de los formatos, será un apuntador a su descripción.

La segunda parte es variable y depende del último campo de la primera. En ella se almacena información referente a parámetros, dimensiones o campos de estructuras.

A continuación presentamos los descriptores de cada una de las estructuras de datos que se pueden declarar en el lenguaje. Los números corresponden a los códigos que se les asignaron a los atributos.

Constante
 Primera
     1) Real (1) o Texto (2)
     2) Constante (3)
     3) Localidad en memoria (tipo Texto)
     4) Uno si es Real, cero si es Texto
 Segunda
     1) Valor de la constante en el caso
        de ser Real.
 del 2 al 4 vacio

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

Arreglo
 Primera
     1) Real (1) o Texto (2)
     2) Arreglo Renglòn (16) Columna o màs dimensiones
        (17). Parámetros por referencia: (18) y (19)
        respectivamente.
     3) Localidad en memoria o localidad en memoria que
        tiene la dirección de la estructura si es
        parámetro
     4) Número de dimensiones (si es un parámetro se
        pondrá un uno)
 Segunda (para cada dimensiòn)
     1) Valor del límite inferior
     2) Valor del límite superior
   3 y 4 vacíos. Si es un parámetro, en el campo 1 se
pondrá el número de las dimensiones y los demàs campos
quedaràn vacíos.

Gràfica
  Primera
     1) Gràfica (3)
     2) Gràfica (17) parámetro por referencia (8).
     3) Cero. En el caso de ser un parámetro, localidad
        en memoria que tiene la dirección de la
        estructura
     4) Seis. Si es un parámetro cero.
  Segunda
     Los campos TITULO, TITCOL y TITREN
     1) Texto (2)
     2) Var. simple (4)
     3) Localidad en memoria
     4) Cero

Campo CUERPO
1) Real (1)
2) Cuerpo Gràfica (10)
3) Localidad en memoria
4) Cero
Dimensiones
1) Valor del primer nûmero o constante en la declaraciòn
2) Valor del segundo nûmero o constante en la declaraciòn
3 y 4 vacios
Màximos y Mínimos de abscisas y ordenadas:
Localidades de
1)Max. abscisas
2)Min. abscisas
3)Max. ordenadas
4)Min. ordenadas

Tabla
Primera
1) Tabla (5)
2) Tabla (25) paràmetro por referencia (26)
3) Cero. Si es un paràmetro, localidad en memoria que tiene la direcciòn de la estructura
4) Nueve. Cero en el caso de ser paràmetro
Segunda
TITULO, TITCOL, TITREN, igual que en la estructura Gràfica.
Campo CUERPO
1) Real (1)
2) Cuerpo Tabla (13)
3) Localidad en memoria
4) Cero
Dimensiones igual que en la estructura anterior.
Campos NOMREN y NOMCOL
1) Texto (2)
2) Arreglo de una o màs dimensiones (17)
3) Localidad en memoria
4) Cero
Campos MARCOREN y MARCOCOL
1) Real (1)
2) Arreglo de una o màs dimensiones (17)
3) Localidad en memoria
4) Cero

Histograma
Primera
1) Histograma (7)
2) Histograma (28) paràmetro por referencia (29)
3) Cero. En el caso de ser paràmetro, localidad donde se encuentra la direcciòn de la estructura.
4) Cuatro. En el caso de ser paràmetro, cero

Segunda
TITULO y NOMREN igual que en la estructura anterior
Campo Cuerpo
1) Real (1)
2) Cuerpo Histograma (22)
3) Localidad en memoria
4) Cero
Dimensiòn
1) Número de las frecuencias
Del 2 al 4 vacio

Archivo
Primera
1) Archivo (10)
2) Cero
3) Nombre interno
4) Cero

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

Rutina
Primera
1) Sin tipo (13) con tipo (12)
2) Rutina (31)
3) Nombre interno
4) Número de parámetros (si es con tipo, se agregará uno más -al principio- con el tipo de la rutina el cual es Real. En la tercera posiciòn se pondrá la localidad donde se dejará el valor de la rutina)
Segunda
Parámetros
1) Tipo
2) Forma
3) Localidad en memoria de la direcciòn de la estructura para parámetros pasados por referencia.
4) Número de dimensiones en el caso de arreglos.

En la figura 4.7 podemos ver unos ejemplos.

FIGURA 4.7

Los Descriptores de las variables de la figura 4.5 son:

```
FUN NIV: (APUNT=  52  IAPDIS=   5 INFO(IAPDIS)=  36
DESCRI: IAPDES=  35 IAPUNT=  52 IP=   1 IS=   4 IT=   40 IC=   0  I
FUN NIV: (APUNT=  53  IAPDIS=   6 INFO(IAPDIS)=  41
DESCRI: IAPDES=  40 IAPUNT=  53 IP=   1 IS=   4 IT=   41 IC=   0  YY
FUN NIV: IAPUNT=  54  IAPDIS=   7 INFO(IAPDIS)=  46
DESCRI: IAPDES=  45 IAPUNT=  54 IP=   1 IS=   4 IT=   42 IC=   0  I
FUN NIV: (APUNT=  55  IAPDIS=   8 INFO(IAPDIS)=  51
DESCRI: IAPDES=  50 IAPUNT=  55 IP=   1 IS=   4 IT=   43 IC=   0  FINARCHIVO
```

IAPVDT es la localidad en Diccionario

IAPDES es el tope de la pila antes de insertar el descriptor

INFO(IAPDIS) es la localidad donde se ubicará el descriptor en la pila

IP, IS, IT e IC son los campos del descriptor

En la figura 4.8 presentamos una parte del programa que genera estos descriptores con el fin de ejemplificar la estructura e implementación del algoritmo que sigue los àrboles de derivación.

FIGURA 4.8

```
9999    CONTINUE
        CALL LEEAN
        IAR=2
        IAAUX=IA(IAR)
        GO TO (1,2), IAAUX
1       IAR=IAR+1
        IAAUX=IA(IAR)
C               D E C L A R A C I O N E S
        GO TO (10,11,12,13,14,15,16,17,18,19),IAAUX
C
C               REALES Y TEXTOS
C
11      CONTINUE
12      IPR=IA(IAR)-1
125     IAR=IAR+1
        IF(IA(IAR).EQ.2)GO TO 9999
        IAR=IAR+1
        IERR=0
        CALL PONNIV(IA(IAR),IERR)
        IF(IERR.NE.0)GO TO 9998
        IAPUNT=IA(IAR)
        IF(IPR.EQ.1) GO TO 127
        ITER=ILOCRE
        ILOCRE=ILOCRE+MAXCHAR/2
        GO TO 128
127     ITER=ILOCRE
        ILOCRE=ILOCRE+1
128     CALL DESCRI(IAPUNT,IPR,4,ITER,0)
        GO TO 125
```

La rutina LEEAN obtiene el árbol de derivación

IAR es un apuntador el árbol, el cual está en el arreglo IA

Se reconoce una declaración o una proposición

Supongamos que se reconoce una declaración de una variable de tipo Real (etiqueta 11)

IPR primer campo del descriptor

Mientras la lista de identificadores no sea vacía

Verifica que no este declarado y si no lo está, prepara las estructuras para incluir el descriptor en la pila (PONNIV)

TEXTO

REAL

Incluye el descriptor en la pila (DESCRI)

4.4.3 Generaciòn de Còdigo

La estructuras descritas en la secciòn anterior tienen como finalidad auxiliar a la generaciòn de còdigo. Esta parte del compilador se encuentra muy ligada al anàlisis semàntico y ambas se desarrollan en forma paralela.

Con el objeto de ilustrar el procedimiento a seguir para el desarrollo de esta parte del compilador se decidiò describir como se generaria el còdigo para la proposiciòn de asignaciòn y la proposiciòn condicional.

Veamos primeramente como se representan cada una de las estructuras en el programa objeto y una descripciòn de las rutinas que se agregaron al lenguaje objeto con la finalidad de ampliarlo y asi facilitar la generaciòn de còdigo.

4.4.3.1 Representaciòn en el programa objeto de las estructuras.

Los valores de todos los objetos manejados en el programa fuente asi como los resultados temporales, resultados de evaluaciòn de expresiones, e informaciòn de control sobre la estructura de bloques son almacenados en el arreglo que llamamos Memoria en el programa objeto. En especial las localidades son indices a dicho arreglo, conformadas por un desplazamiento fijo almacenado en el descriptor, y un valor base guardado en el registro de despliegue (display) del nivel lexicogràfico correspondiente.

Veamos cual seria el equivalente de cada una de las estructuras en el programa objeto.

Una constante de tipo Real se sustituye en el programa objeto por su valor el cual se toma del descriptor.

Para la representaciòn en el programa objeto de una variable de tipo Real se utiliza una celda de la Memoria. Su localidad se encuentra en el descriptor. Para el caso de paràmetros formales pasados por referencia se asume que el valor que se encuentra en la localidad referida en el descriptor es la direcciòn de la variable. Cuando esta variable es requerida, se maneja indirectamente. En el llamado a la rutina, los paràmetros actuales se depositan en el tope de la Memoria, donde son referenciados por los descriptores de los paràmetros formales. Los paràmetros pasados por valor se manejan como variables locales cuyo valor

inicial es el valor del parámetro actual.

Una variable, al igual que una constante de tipo Texto ocupará 30 localidades de Memoria donde en cada localidad se almacenarán dos caracteres.

El restringir a una longitud fija a las variables de tipo Texto, facilita en cierta forma su manejo, aunque nos restringe a cadenas de caracteres de una longitud menor o igual a 60. Considerando la orientación del lenguaje se decidió manejar este tipo de variables con longitud fija.

Los arreglos de tipo Real y las expresiones cuyos valores fuesen de tipo Arreglo se manejaron de la siguiente forma:

La localidad a la que se hace referencia en el descriptor contiene el número de dimensiones del arreglo. En el caso de ser uno, para diferenciar entre renglón y columna, se pondrá un 1 para columna y un -1 para renglón. Si este número es N, las |N| posiciones que sigan serán ocupadas por las cardinalidades de cada dimensión, las siguientes localidades de Memoria, serán para los valores los cuales ocuparán una posición. Así pues, se llevará un descriptor de estas estructuras en ejecución.

Ejemplo

2	3	2	10	20	30	40	50	60

Será un arreglo de dos dimensiones con tres renglones y dos columnas. Los valores de cada renglón serán: 10 20, 30 40 y 50 60.

El no guardar el límite inferior y superior de los arreglos en su representación en ejecución no presenta problemas pues el descriptor contiene estos valores. Cuando el compilador genere código para evaluar un índice, generará código también para que al resultado de la expresión se le reste la expresión (límite inferior - 1). En forma parecida se puede checar si un índice es inválido.

Para el caso de las expresiones cuyo resultado sea de tipo Arreglo, el límite inferior se asume que es uno como se estableció en la sección 3.3. La representación de los arreglos de tipo Texto, es similar a los de tipo Real con la diferencia de que los valores ocuparán 30 celdas de Memoria.

Los campos de las estructuras Gráfica, Tabla e

Histograma se representan de la siguiente manera:

TITULO, TITCOL y TITREN como variables de tipo Texto; NOMREN y NOMCOL como arreglos de tipo Texto; el cuerpo de una Gràfica, de una Tabla y de un Histograma se representan como arreglos reales al igual que MARCOCOL y MARCOREN; los màximos y mínimos de abscisas y ordenadas de una estructura Gràfica, como variables de tipo Real.

Una vez descrita la representación de cada una de las estructuras en el programa objeto, analicemos màs detalladamente nuestro código objeto.

4.4.3.2 Ampliaciones al lenguaje objeto.

Con el fin de facilitar la generación de código y màs aun, para que los algoritmos que se utilicen para evaluar las expresiones pudiesen ser fàcilmente modificados, las expresiones se traducen a una serie de llamados a rutinas en Fortran.

Estas rutinas siguen bàsicamente la misma lógica en cuanto a cómo deben recibir los operandos con los que trabajaràn y cómo deben dejar el resultado èsto con el fin de uniformizar el procedimiento y que de esta forma sea fàcil eventualmente enriquecer el código objeto con nuevas rutinas.

Se aumentó al lenguaje objeto una rutina para cada una de las operaciones unarias y binarias. Estas últimas tienen la siguiente forma:

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

Donde "OPERACIONBINARIA" es el nombre de la rutina para cada una de las operaciones (suma, resta, multiplicación, división, exponenciación, multiplicación de matrices, y lógico, o lógico y operaciones de relación).

"LOC1", "LOC2" y "LOC3" son las localidades en Memoria del primer operando, segundo operando y la localidad donde se dejarà el resultado respectivamente.

"TIPO1", "TIPO2" y "TIPO3" identifican como se deberà interpretar la información ubicada en las distintas localidades de memoria. Llamaremos a esta interpretación el "tipo en ejecución". Los posibles valores son: 1 = Real, 2 = Arreglo Real, 3 = Texto, 4 = Arreglo Texto. Estos valores corresponden a todas las representaciones descritas en 4.4.3.1.

Obsèrvese que en tiempo de ejecuciòn, ùnicamente con saber la localidad de un objeto y còmo interpretar su representaciòn basta para poder manipularlo. Esto se logra con los paràmetros "LOC" y "TIPO".

De los seis paràmetros el ùltimo lo determina la rutina, mientras que los demàs los recibe.

Las operaciones con un solo operando como son traspuesta, inversa, determinante, negaciòn y menos unario, tienen la siguiente forma:

OPERACIONUNARIA(LOC1,TIPO1,LOC2,TIPO2)

Donde "OPERACIONUNARIA" es el nombre de la rutina. "LOC1" y "LOC2" son las localidades del operando y el resultado respectivamente. "TIPO1" y "TIPO2" son el tipo en ejecuciòn del operando y el resultado respectivamente.

Estas rutinas dejaràn sus resultados temporales a partir del tope de la Memoria.

La operaciòn de Dimensionar se llevarà a cabo al no tomar en cuenta en las rutinas las dimensiones con un solo elemento.

Se aumentaron tambièn las rutinas siguientes:

METE(LOC,TIPO) SACA(LOC,TIPO)

Estas rutinas se hicieron con el objeto de administrar una pila de operadores en ejecuciòn al evaluar las expresiones. La primera mete en el tope de la pila la localidad y el tipo en ejecuciòn de un operando determinado y la segunda hace la operaciòn inversa. En esta pila se guardan las localidades y los tipos en ejecuciòn de los resultados temporales. Esta pila se usa debido a que el nùmero de celdas en Memoria que ocupa un operando no es constante.

Otra rutina que se aumentò al lenguaje objeto con el fin de poder mover informaciòn en la memoria y para resolver la generaciòn de còdigo para la <EXPRESION SELECTIVA DE ARREGLO> fue la siguiente

MUEVE(LOC1,TIPO1,SELEC1,LOC2,TIPO2,SELEC2)

Donde "LOC1" y "TIPO1" describen la estructura de donde se tomarà la informaciòn. "SELEC1" es un vector que se utiliza en el caso de que "TIPO1" sea 2 (Arreglo Real) o 4 (Arreglo Texto) donde se indican los elementos que se tomaràn de las dimensiones. "LOC2" y "TIPO2"

indican la localidad donde se ubicarà la informaciòn y la forma como quedarà representada. "SELEC2" indica, en el caso de arreglos, en que parte del arreglo quedarà ubicada la informaciòn. Si el paràmetro "TIPO2" no es informado la rutina se encargarà de ponerlo.

Una forma de representar "SELEC1" y "SELEC2" es la siguiente:

Los elementos de cada dimensiòn se indicaràn a travès de una serie de parejas de nùmeros donde cada pareja indica de cuàl elemento a cuàl otro elemento se tomarà. Si en una dimensiòn determinada apareciese la pareja (-1,0) significaria que los elementos de esa dimensiòn se sumarian; si apareciese la pareja (-2,0) indicaria que se tomarian todos los elementos de esa dimensiòn.

Ejemplo:

Si tuvièsemos la siguiente representaciòn interna de un arreglo Real

localidad	n	n+1	n+2	n+3 ...	n+62
valores	2	6	10	x1 ...	x60

Donde x1...x60 son los valores del arreglo, los valores de "LOC1" y "TIPO1" serian: "LOC1" = n, "TIPO1" = 2.

Si los valores de "SELEC1" fuesen:

2	1	3	5	3	1	2	4	7	9	9

Significaria que el arreglo se compone de 2 dimensiones (por el primer 2), en la primera dimensiòn, hay una pareja (por el primer 1) que indica que se tomaria del elemento 3 al 5 (por los nùmeros 3 y 5), y en la segunda dimensiòn hay tres parejas (por el segundo 3) que indican que se tomaràn del elemento 1 al 2, del 4 al 7 y el 9. Corresponderia a la siguiente <EXPRESION SELECTIVA DE ARREGLO> :

[3 _ 5 , 1 _ 2 & 4 _ 7 & 9]

Si los valores fuesen

2	1	-1	0	1	-2	0

Significaria que el arreglo se compone de dos dimensiones (por el primer 2), en la primera dimensiòn hay una pareja que indica que los elementos de esa dimensiòn se sumarian (-1,0) y en la segunda dimensiòn

hay una pareja que indica que se tomarían todos los elementos (-2,0). Corresponderia a la siguiente <EXPRESION SELECTIVA DE ARREGLO> :

```
[ . , ]
```

Por último, tendríamos una rutina que dada la localidad de una variable o estructura y su tipo en ejecución, regresaría su longitud:

```
LONG(LOC1,TIPO1,ILONG)
```

Recibe "LOC1 que es la localidad y "TIPO1" su tipo en ejecución y entrega en la variable ILONG el número de caracteres que ocupa esa estructura.

Con estas rutinas el generar código para las expresiones cuyo valor sea de tipo Real o Arreglo Real al igual que para la asignación, no resulta ser difícil.

Veamos un ejemplo:

Recordemos la expresión de la sección 3.3 para calcular la suma de la edades entre 5 y 90

```
SUMA:=(((EDADES>5)Y(EDADES<90))*EDADES)[ . ]
```

Suponiendo que la localidad de SUMA fuese 101 y la de EDADES fuese 102, el código generado para esta asignación sería el siguiente:

```
1    LOC1=DISPLAY(NIVEL) + 102
2    MEMORIA(TOPEMEM)=5
3    LOC2=TOPEMEM
4    TOPEMEM=TOPEMEM+1
5    CALL MAYQUE(LOC1,2,LOC2,1,TOPEMEM,TIPO3)
6    CALL METE(LOC3,TIPO3)
7    CALL LONG(LOC3,TIPO3,ILONG)
8    TOPEMEM=TOPEMEM+ILONG
9    LOC1=DISPLAY(NIVEL) + 102
10   MEMORIA(TOPEMEM)=90
11   LOC2=TOPEMEM
12   TOPEMEM=TOPEMEM+1
13   CALL MENQUE(LOC1,2,LOC2,1,TOPEMEM,TIPO3)
14   CALL METE(TOPEMEM,TIPO3)
15   CALL LONG(LOC3,TIPO3,ILONG)
16   TOPEMEM=TOPEMEM+ILONG
17   CALL SACA(LOC2,TIPO2)
18   CALL SACA(LOC1,TIPO1)
```

```
19   LOC3=LOC1
20   CALL CONJY(LOC1,TIPO1,LOC2,TIPO2,LOC3,TIPO3)
21   CALL METE(LOC3,TIPO3)
22   CALL LONG(LOC3,TIPO3,ILONG)
23   TOPEMEM=TOPEMEM+ILONG
24   CALL SACA(LOC1,TIPO1)
25   LOC2=DISPLAY(NIVEL) + 102
26   LOC3=LOC1
27   CALL MULT(LOC1,TIPO1,LOC2,2,LOC3,TIPO3)
28   CALL METE(LOC3,TIPO3)
29   CALL LONG(LOC3,TIPO3,ILONG)
30   TOPEMEM=LOC3+ILONG
31   CALL SACA(LOC1,TIPO1)
32   SELEC1(3)=-1
33   SELEC1(4)= 0
34   SELEC1(2)= 1
35   SELEC1(1)= 1
36   LOC2=DISPLAY(NIVEL) + 101
37   CALL MUEVE(LOC1,TIPO1,SELEC1,LOC2,1,SELEC2)
38   TOPEMEM=LOC1
```

1 a 4	Se preparan los parámetros para la operación binaria MAYQUE (mayor que).
1	Esta es la localidad de EDADES que se obtiene del descriptor. DISPLAY es un arreglo con las localidades absolutas de los niveles.
2,3	La constante 5 se coloca en el tope de la Memoria.
4	Se actualiza el tope de la Memoria
5	Se llama a la rutina que ejecuta la operación mayor que. El tipo en ejecución de EDADES se deduce del descriptor. El tipo en ejecución de la constante se puede deducir facilmente al reconocer que el operando es un número. Si el operando hubiese sido un <ARREGLO EXPLICITO> el tipo hubiera sido 2 y se hubiera generado código para colocar al arreglo en el tope de la Memoria, como se puede observar la disposición de los elementos del <ARREGLO EXPLICITO> es la misma de la representación de los arreglos en Memoria
6	Se guarda la localidad y el tipo en ejecución del resultado temporal en la pila de operandos.
7,8	Se actualiza el tope de la Memoria
9 a 12	Se preparan los parámetros para la operación binaria MENQUE (menor que)
13	Se realiza la operación
14	Se guarda la localidad y el tipo en ejecución del resultado temporal en la pila de operandos. Si se quiere optimizar el código, este paso puede omitirse pues a continuación (17) se saca esta información

15,16	Se actualiza el tope de la Memoria
17 a 20	Se preparan los parámetros para la operación binaria CONJY (conjunción Y) y se realiza la operación
21 a 23	Se actualizan el tope de la Memoria y la pila de operandos
24 a 26	Se preparan los parámetros para la operación binaria MULT (multiplicación)
27	Se realiza la operación
28 a 30	Se actualiza el tope de la Memoria y la pila de operandos
31 a 36	Se preparan los parámetros para la operación MUEVE para realizar la asignación
31	Se actualizan los parámetros LOC1 y TIPO1 con el resultado de la expresión
32,33	Primera pareja de la primera dimensión. En este caso se indica la suma de los elementos de esa dimensión
34	Número de parejas de la primera dimensión
35	Número de dimensiones
36	La localidad y el tipo en ejecución de la variable SUMA son tomados del descriptor
37	Se realiza la asignación
38	Se actualiza el tope de la Memoria

Para la <PROPOSICION CONDICIONAL> únicamente necesitamos generar saltos incondicionales, saltos condicionados y etiquetas seguidas de la proposición de control "CONTINUE".

Veamos un ejemplo:

Para la siguiente proposición

```
SI !A! <> 0 ENTONCES
  X:=A@><C
SINO
  ESCRIBE EN ARCH CON "EL DETERMINATE",
    " ES IGUAL A CERO"
```

El código generado para la <PROPOSICION CONDICIONAL> sería el siguiente asumiendo que la rutina que evalua la expresión !A! <> 0 dejó el resultado en una celda de Memoria cuya localidad se encuentra en LOC3:

```
1          IF(MEMORIA(LOC3).EQ.0)GO TO 9001

           Código para la asignación

2          GO TO 9002
3    9001 CONTINUE
```

Còdigo para la prop. de escritura

```
4    9002 CONTINUE
```

1 Se genera un salto condicionado. Recordemos que si el resultado de la expresiòn es cero se efectuarà la alternativa (secciòn 3.4.5)

2. Una vez generado el còdigo para la asignaciòn se genera un salto incondicional a una etiqueta que estarà al final de la proposiciòn

3 Se genera una etiqueta donde se pondrà el còdigo de la alternativa. Esta debe ser a la que se hace referencia en (1)

4 Despuès de generar el còdigo para la proposiciòn de escritura, se genera la etiqueta referenciada en (2)

4.4.3.3 Descripciòn de la generaciòn de còdigo para la proposiciòn de asignaciòn y la proposiciòn condicional

El compilador administrarà una pila de operandos y operadores para generar el còdigo para las operaciones.

Cada operador tendrà un còdigo asociado el cuàl se usarà para identificarlo.

Este còdigo serà el que se use en la pila de operadores. Al mismo tiempo el còdigo asociado se utilizarà para facilitar el manejo de la precedencia de los operadores. El "(" tiene precedencia diferente al entrar a la pila que yà una vez en ella. Al entrar siempre debe quedarse en el tope (precedencia màxima). Una vez en la pila su precedencia debe ser mìnima para que se aparee con ")".

Para los operadores se almacenarà en la pila su descriptor para de ahí tomar las caracterìsticas que se requieran como localidad y tipo en ejecuciòn. Si los operadores son resultados temporales de expresiones, se indicarà si las caracterìsticas de la estructura se tomaràn de la pila en ejecuciòn (con la rutina SACA).

Al identificarse un operador en el àrbol de derivaciòn, la precedencia de èste se compararà con la del operador que estè en el tope de la pila de operadores, si es que hubiese un operador en la pila.

Segùn el resultado de esta comparaciòn, el operador se meterà en la pila o se generarà còdigo para efectuar

la operaciòn con el o los operandos que estèn en el tope de la pila de operandos. En la pila de operandos se indicarà si el operando es temporal o no con el objeto de poder saber si las celdas de Memoria que ocupa pueden utilizarse nuevamente.

Para la asignaciòn se generarà còdigo para que el resultado se ponga en la <ESTRUCTURA REFERENCIABLE> (con la rutina MUEVE).

En cuanto a la proposiciòn condicional, una pila de etiquetas es necesaria para generar el còdigo respectivo.

Al identificarse un "SI", se genera una etiqueta la cuàl se guarda en la pila y se genera còdigo para saltar a ella en caso de que la expresiòn sea igual a cero. En esta etiqueta (etiqueta 1) se ubicarà la alternativa o el fin de la proposiciòn en su caso.

Un "INICIO" despuès del "SI...ENTONCES" provocarà que antes de generar la etiqueta, se marque primero el tope de la pila. El "INICIO" harà las veces del parentesis abierto en el caso de las expresiones, esto con el fin de considerar proposiciones condicionales dentro de proposiciones compuestas.

Si existe alternativa se genera una etiqueta (etiqueta 2) para saltar a ella incondicionalmente y ademàs el salto respectivo que brinca la alternativa, esta etiqueta se ubicarà al final de la proposiciòn. En seguida se genera còdigo para escribir en el programa objeto la etiqueta 1 la cuàl se encuentra en el tope de la pila. A continuaciòn se retira èsta de la pila y se mete la etiqueta 2. En seguida se pondrà el còdigo de la alternativa. Al final de esta ùltima se pone la etiqueta 2 en el programa objeto.

5. EXTENSIONES FUTURAS

Sobra decir que el completar las fases del anàlisis semàntico y generaciòn de còdigo es una condiciòn necesaria para que se puedan ejecutar programas en nuestro lenguaje. Sin embargo, con el trabajo realizado las bases estàn sentadas para continuar la labor de implementaciòn del compilador.

A continuaciòn se sugieren dos ideas que podrian ayudar al enriquecimiento del compilador tomando en cuenta su orientaciòn y el hecho de que pueda seguir implementàndose en una minicomputadora.

5.1 Rutinas Externas

Enriquecer el lenguaje con la posibilidad de incluir en un programa llamados a rutinas externas escritas en lenguaje objeto (Fortran) permitiria que el usuario pudiese incluir rutinas hechas por èl mismo y tambièn el poder utilizar todas la rutinas predefinidas en el lenguaje Fortran.

Para incluir esta extensiòn se sugiere lo siguiente:

Primeramente dotar al lenguaje con una declaraciòn en la que se especifique el nombre de la rutina y sus paràmetros:

```
<DECLARACION DE RUTINA EXTERNA>::=
      EXTERNA <IDENTIFICADOR>=<TEXTO>
       <ZONA DE PARAMETROS>
```

Donde <IDENTIFICADOR> seria el nombre interno de la rutina y en el <TEXTO> iria el nombre externo de la misma.

Esto implicaria incorporar esta nueva regla sintàctica, crear una nueva palabra reservada (EXTERNA) y asociarle una producciòn del tipo B() con el número asociado a la misma.

El descriptor de esta rutina seria parecido al de la rutina sin tipo. Se asignaria un nuevo tipo (primera posiciòn del descriptor), y en el lugar del nombre de la rutina (tercera posiciòn del descriptor), se pondria un apuntador a la tabla de caracteres donde estaria el nombre externo.

La sintaxis del llamado a una rutina no tiene por què ser distinto al llamado a una rutina sin tipo. Asi

mismo, el propio compilador checaría que los tipos entre parámetros formales y actuales coincidiesen.

En cuanto al código generado, éste sería un llamado a la rutina cuyo nombre externo se tomaría de la tabla de caracteres y por cada parámetro formal se generaría en la zona de parámetros una pareja de variables conteniendo la localidad del parámetro actual en la Memoria y su tipo en ejecución.

La rutina construida por el usuario deberá primeramente considerar que por cada parámetro en el llamado a la rutina en nuestro lenguaje, deberá poner dos parámetros formales en la rutina en lenguaje objeto (Fortran) siendo el primero la localidad donde está el parámetro actual y el segundo el tipo en ejecución; así mismo se tomará en cuenta como se almacenan las distintas estructuras en Memoria.

Ejemplo:

Supóngase que se tiene una subrutina en Fortran que genera números aleatorios a partir de una semilla cuyo encabezado es el siguiente

```
SUBROUTINE ALEAT(SEM)
```

En el programa fuente se declararía de la siguiente forma

```
EXTERNA
 NUMEROALEATORIO="ALEAT"(REAL SEM)
```

y se llamaría así

```
   NUMEROALEATORIO(SEMILLA)
```

donde SEMILLA sería una variable de tipo Real.

El código generado sería

```
CALL ALEAT(LOC,TIPO)
```

Donde LOC contiene la localidad en Memoria de la variable SEMILLA y TIPO su tipo en ejecución (en este caso sería uno por ser de tipo Real)

La subrutina "ALEAT" sufriría las siguientes modificaciones:

1) El encabezado sería
   SUBROUTINE ALEAT(LOC,TIPO)
2) Se agregaría un COMMON para hacer común el área de Memoria
3) En el cuerpo de la subrutina, donde apareciese SEM habría que poner MEMORIA(LOC) donde MEMORIA es el arreglo de la Memoria

5.2 Opciones del Compilador

Otra extensión que se podría incorporar sin mayor problema es la siguiente:

Durante el desarrollo del compilador se hicieron varias rutinas para su depuración como es el escribir el "texto limpio", los árboles de derivación, los descriptores. Se sugiere hacerlas opcionales para depurar los programas fuente. Se podría incluir al principio del programa un comentario opcional donde se especifiquen las opciones que se quieran como se hace en ciertas implementaciones de Pascal,Fortran, Cobol, etc.

6. CONCLUSIONES

El haber llevado este proyecto desde su inicio definiendo los requerimientos con los futuros usuarios, pasando por el diseño del lenguaje hasta llegar a la implementación en que se encuentra actualmente, me permitió pasar por algunas de las distintas fases que se siguen durante el desarrollo de un lenguaje.

Es satisfactorio ver que estas experiencias no son propias de un proyecto de esta naturaleza sino que son aplicables en el desarrollo de cualquier sistema, o cualquier problema que requiera de una solución.

Entre estas experiencias quisiera resaltar las siguientes:

El definir los requerimientos del lenguaje requirió de una labor de equipo. El extraer de un conglomerado de necesidades las realmente importantes y el entender el problema que se deseaba resolver requirió una serie de pláticas con los futuros usuarios, consultar bibliografía proporcionada por ellos mismos respecto a los problemas a los que se enfrentaban y un análisis de soluciones ya existentes.

La aprobación del diseño del lenguaje por parte de los profesores de estadística del Laboratorio de Estadística de la Facultad de Ciencias coronó este trabajo.

Es importante compenetrarse en el problema para que la solución que se proponga sea utilizada.

En cuanto a la implementación, el haber trabajado con una minicomputadora dificultó el problema. La solución que se tenía que desarrollar estaba sujeta a los recursos de la máquina tanto en lo que respecta a los lenguajes utilizados en los programas como en lo que respecta a la capacidad de la memoria.

A lo largo del desarrollo, se tuvo muy en cuenta que se quería implementar en una minicomputadora.

Características como son el manejo de arreglos dinámicos, el que las rutinas con tipo regresaran como valor toda clase de tipos, el manejo de parámetros tipo arreglo llamados por valor, el manejo de cadenas de caracteres de longitud variable, entre otras, tuvieron que dejarse a un lado para lograr una implementación adecuada a los recursos. A cambio de esto se trabajó

para que el manejo de estructuras, como las matrices, se facilitara en las expresiones. Asi mismo se facilitò el manejo de estructuras como gràficas, tablas e histogramas.

El resultado fue un lenguaje tanto de propòsito general como un lenguaje con una orientaciòn al tratamiento de problemas estadisticos.

Esta misiòn dual del lenguaje lo hace ser una aportaciòn tanto al campo del diseño de lenguajes de programaciòn como al campo de la estadistica.

Una de las caracteristicas màs importantes tanto del diseño del lenguaje como de la implementaciòn es su modificabilidad. Esta caracteristica siempre se tuvo presente precisamente para propiciar la continuaciòn del proyecto por parte de estudiantes que en el futuro deseasen elaborar trabajos basados en el presente.

En ocasiones se optò por concesiones negativas en eficiencia, por considerarse màs importante cubrir una gama amplia de diferentes aspectos del proyecto, que abocarse a la implementaciòn de algoritmos eficientes, conocidos y apremiantes en cuanto a tiempo de desarrollo.

Por último, y no por esto menos importante, quisiera hacer notar lo siguiente: Al leer programas escritos en este lenguaje las palabras utilizadas me parecieron un tanto extrañas por estar en español. Esto me hizo pensar en el grado de dependencia en que se encuentra la computaciòn en general en nuestro país. Pràcticamente todos los lenguajes computacionales se encuentran al alcance de minorias debido en gran parte a que estàn en inglès. Asi mismo nos resulta extraño leer textos de cualquier campo de la computaciòn en nuestro idioma. Con este trabajo deseo contribuir un poco a la labor desarrollada en contra de la dependencia en que nos encontramos.

APENDICE A

LA CONFIGURACION DEL EQUIPO DE COMPUTO INSTALADO EN EL LABORATORIO DE LA FACULTAD DE CIENCIAS ES LA SIGUIENTE:

MODELO	DESCRIPCION.
8408-H	Procesador Central Nova 3/12 con 65536 bytes memoria MOS.
8530	Cargador automático de programa.
8531	Monitor de alimentación
8532	Respaldo de baterias para memoria MOS.
4007	Interfase E/S
4008	Reloj de tiempo real
4010	Control de consola
6040-DA	Consola de Operación
6030	Subsistema de Diskettes.
1012-K	Gabinete.

LOS PROGRAMAS BASICOS DE ESTE EQUIPO INSTALADO SON:

PROGRAMA	DESCRIPCION
1. DOS	Disk Operating System, para sistemas de diskettes con programas de utileria
2. FORTRAN IV	Compilador para la configuración del equipo elegido.
3. BASIC	Interprete Basic configurado para el equipo elegido

APENDICE B
MANUAL DE REFERENCIA Y OPERACION

El propòsito de este manual es describir el funcionamiento y ubicaciòn de la implementaciòn del compilador y los programas de apoyo. Tambièn se describe la manera de operar los programas. Esta implementaciòn corresponde a la que se hizo en el Laboratorio de Estadistica de la Facultad de Ciencias. Los discos flexibles a los que se haga referencia se localizan en dicho lugar.

El compilador se encuentra dividido en dos grandes partes: El analizador lexicogràfico y sintàctico por un lado y el analizador semàntico por otro. Los programas fuente y objeto de los primeros se encuentran en el disco marcado con la etiqueta "Lèxico Sintàctico". Los del analizador semàntico se encuentran en el disco marcado con la etiqueta "Semàntico". Los archivos donde estàn los programas fuente se distinguen de los objeto por el sufijo que el sistema le pone a los archivos con còdigo objeto (.RB binario relocalizable). Los fuente no tienen sufijo.

Los programas de apoyo se encuentran en el disco "Lèxico Sintàctico".

Dividiremos nuestra descripciòn en:

1) Programas de Apoyo
2) Analizador Lexicogràfico y Sintàctico
3) Analizador Semàntico
4) Operaciòn

Nota: Los nombres de los archivos que tienen programas, funciones, subrutinas y los archivos que contienen la gramàtica y las palabras reservadas aparecen en los discos con una terminaciòn numèrica (Ejemplo: LEX26). Este nùmero se incluyò para identificar la versiòn durante el desarrollo. En las siguientes lìneas cuando nos refiramos a estos archivos no incluiremos el nùmero, en lugar de èste se pondrà el nùmero cero (Ejemplo: LEX0), pero el nùmero real se requiere para cuando quiera uno referirse a dichos archivos. Listando los directorios de los discos se pueden obtener las terminaciones numèricas correspondientes (comando LIST de CLI [D1]).

1. Programas de Apoyo

La función de estos programas es crear la gramàtica interna y el archivo con el que se inicializarà el Diccionario y la Tabla de Caracteres con las palabras reservadas.

Estos programas fueron escritos en Basic por las facilidades en cuanto a manejo de caracteres del Basic Extendido de la NOVA 3/12[D21.

>>>Programa: HTO

Archivo : LJGGHTO

Descripción:
Este programa recibe una gramàtica escrita en el metalenguaje descrito en la sección 4.3.1 y obtiene la gramàtica interna equivalente

La gramàtica debe encontrarse en un archivo llamado LJGGGTOO (este archivo fue el que se listò para la sección C.1 del apèndice C).

Al ejecutarse el programa se genera un archivo llamado LJGGGRAMGEN (sección C.2 del apèndice C) que contiene la gramàtica interna y otro, LJGGDIAGNOS (sección C.3 del apèndice C) que contiene la ubicación de cada uno de los simbolos no terminales en la tabla gramatical. Si algún simbolo no estuviese definido, aqui se detectaria el error.

Archivos de entrada:
LJGGGTOO

Archivos de salida:
LJGGGRAMGEN
LJGGDIAGNOS

>>>Programa: HPCO

Archivo : LJGGHPCO

Descripción:
Este programa toma un listado de las palabras reservadas y traduce cada palabra a un número, que es el número de caracteres de que se compone, y los códigos ASCII de cada una de sus letras para ser cargadas posteriormente a la tabla de caracteres. Asi mismo, les asigna una dirección en el Diccionario.

Las palabras reservadas deberàn encontrarse en un archivo llamado LJGGPALCLO, la traducciòn de las palabras se ubicarà en un archivo llamado LJGGPCCHAR (secciòn C.5 del apèndice C) y sus respectivas localidades en el Diccionario en LJGGPCVDT.

Se genera asi mismo un listado de las palabras reservadas relacionadas con su direcciòn correspondiente en LJGGRELPC (secciòn C.4 del apèndice C)

Archivos de entrada:
LJGGPALCLO

Archivos de salida :
LJGGPCCHAR
LJGGPCVDT
LJGGRELPC

2 Analizador Lexicogràfico y Sintàctico

Esta parte del compilador se compone de un programa principal (archivo: LEXO) ocho subrutinas y cinco funciones (archivos: BUSCO, ERRORO, GUARDAO, ISIGNBO, ISIGSIO, LEESIMO, PARSEO, PONCHSO, POPO, PRTARO, PUSHO, VEELNBO).

>>>Programa: (MAIN - por sistema)

Archivo : LEXO

Descripciòn:
Este programa recibe como entrada la gramàtica interna (LJGGGRAMGEN), la informaciòn referente a las palabras reservadas para cargar la Tabla de Caracteres (LJGGPCCHAR) y el Diccionario (LJGGPCVDT) y el programa fuente (PROG) generando a su vez los àrboles de derivaciòn (ANREC) y la Tabla de Caracteres con el Diccionario (CHARVDT) para el analizador semàntico. Asi mismo, genera un listado del programa marcando los errores sintàcticos encontrados ($TTO).

Las subrutinas y funciones màs importantes son:

PARSE (archivo: PARSEO) que se encarga de efectuar el recorrido de la gramàtica para construir los àrboles de derivaciòn.

BUSC (archivo: BUSCO) que se encarga de buscar en el Diccionario los identificadores

PONCHS (archivo:PONCHSO) que incorpora a la Tabla de Caracteres los identificadores, los números y las cadenas de caracteres.

Archivos de entrada:
LJGGGRAMGEN
LJGGPCCHAR
LJGGPCVDT
PROG

Archivos de salida:
ANREC
CHARVDT
$TTO

3.Analizador Semàntico

El Analizador Semàntico se compone de un programa principal (archivo: SEMO) nueve subrutinas y una funciòn (archivos: DESCRIO, ERRORO, GENO, INICIAO, INIGENO, LEEANO, PONIVO, VALORO, VEDESCO, VESIGO)

>>>Programa: (MAIN - por sistema)

Archivo : SEMO

Descripciòn:
Este programa recibe como entrada los àrboles de derivaciòn (ANREC) y la Tabla de Caracteres con el Diccionario (CHARVDT) y entrega como resultado los errores semànticos ($TTO).

Actualmente el Analizador Semàntico es capaz de construir los descriptores de todos los elementos que se declaren llevando las estructuras que se describen en la secciòn 4.4.1.

Las subrutinas màs importantes son:

PONNIV (archivo: PONIVO) que administra las estructuras necesarias para localizar el descriptor de un determinado identificador

VEDESC (archivo: VEDESCO) obtiene el descriptor de un identificador determinado

DESCRI (archivo: DESCRIO) pone el descriptor de un determinado identificador en el tope de la pila de descriptores.

Archivos de entrada:
ANREC

CHARVDT

Archivos de salida:
$TTO

4. Operaciòn.

4.1 Proceso de creaciòn de la gramàtica interna y archivo con que se inicializarà el Diccionario y la Tabla de Caracteres (creando una copia).

a) Copiar en un disco que estè ligado con el intèrprete Basic los archivos LJGGHTO y LJGGHPCO que se encuentran en el disco marcado con la etiqueta "Lèxico Sintàctico". (referirse al manual [D1] para formatear e inicializar un disco y ligarlo con el intèrprete Basic)

b) Copiar la nueva gramàtica al disco con el nombre de LJGGGTOO

c) Ejecutar el programa HTO con la cuenta "LJGG" para crear la gramàtica interna que quedarà en el archivo LJGGGRAMGEN

d) Ejecutar el programa HPCO para crear los archivos LJGGPCCHAR y LJGGPCVDT (inicializaciòn de Diccionario y Tabla de caracteres)

4.2 Proceso para ejecutar el compilador (creando una copia)

a) Copiar en un disco los archivos LEXO.SV y SEMO.SV los cuales se encuentran en los discos marcados con las etiquetas "Lèxico Sintàctico" y "Semàntico". Estos archivos contienen el cargado de los còdigos objeto de las rutinas de los analizadores.

b) Copiar los archivos generados en el proceso descrito en 4.1 o tomar los archivos originales que se encuentran en el disco "Lèxico Sintàctico" (LJGGGRAMGEN, LJGGPCCHAR, LJGGPCVDT)

c) Editar el programa fuente y salvarlo con el nombre de "PROG"

d) Ejecutar LEXO.SV y SEMO.SV (cargado de rutinas)

Los pasos a) y b) se tendràn que ejecutar sòlo una vez para cargar el compilador al disco de trabajo. Posteriormente sòlo los pasos c) y d) son los que se

tendràn que ejecutar.

4.3 Proceso para modificar algùn programa o rutina del compilador.

a) Copiar la versiòn fuente de la rutina a modificar

b) Modificarla y salvarla actualizando el número de versiòn. Ejemplo: Si se quiere modificar GUARDA5, la versiòn modificada deberà llamarse GUARDA6.

c) Compilar la nueva versiòn. Se genera entonces la versiòn objeto correspondiente (con terminaciòn .RB)

d) Crear nuevamente el archivo LEXO.SV y/o SEMO.SV según si se modificò una rutina del analizador Lexicogràfico y Sintàctico y/o Semàntico. Para èsto se copiaràn los archivos con los còdigos binarios relocalizables de las demàs rutinas (terminaciòn .RB).
La creaciòn de LEXO.SV se efectùa de la siguiente forma:

```
RLDR LEXO BUSCO ERRORO ESCCHVO GUARDAO ISIGNBO ISIGSIO
LEESIMO PARSEO PONCHSO POPO PRTARO PUSHO VEELNBO FORT.LB
SYS.LB
```

El comando RLDR asume que las versiones de los archivos que deberà tomar son las que tienen el prefijo .RB. Se pondrà como primer archivo aquèl donde se encuentra el programa principal; en el resto de los archivos estaràn las subrutinas y funciones. Si se llegara a incluir una màs, èsta deberà ponerse en esta lista. Los ùltimos dos archivos FORT.LB y SYS.LB contienen las bibliotecas del compilador y del sistema Fortran.

La creaciòn de SEMO.SV se efectùa de la siguiente forma:

```
RLDR SEMO DESCRIO ERRORO GENO INICIAO INIGENO LEEANO
PONIVO VALORO VEDESCO VESIGO FORT.LB SYS.LB
```

Con esto se tendrà la nueva versiòn de LEXO.SV y SEMO.SV pudièndose continuar con el proceso 4.2

APENDICE C

GRAMATICA Y PRODUCTOS DE APOYO

C.1 Gramática a partir de la cual se generó la gramática interna para el Analizador Sintáctico

```
TYPE LJGGGT027
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(NUM)=1010;
B(IDENT)=1020;
B(TEX)=1040;
B(PI)=2040;
B(E)=2041;
          %********* DECLARACIONES Y PROPOSICIONES      **********%
P(PRINCIPIO)=(P) ';';
```

```
P(P)= (DECL)! (L ETIQ) (P GRAL);
P(P GRAL)=(CUNO) ! (ITER) ! (SELEC)  !(P INC);
P(RF1)=(HAS)!(RC1);
P(L ETIQ)=(NUMERO)':'(L ETIQ)!;
P(P INC)=(PR INC)(TERMINA);
P(TERMINA)=(FIN)(RF1)(TERMINA)!;
P(PR INC)=(ASIG)!(INICIO)(P)!(LLAMADO)!(P VER)!(LECTURA)!(ESCRITURA)!;
P(DECL)=(CTE)(L DE CTES)!(REAL) (L DE ID) !
   (TEXTO)(L DE ID)!(ARREGLO) (TIPO ARR)(L DE ARREGLOS)!
  (GRAFICA) (L DE GRAF TAB) !(TABLA)(L DE GRAF TAB)!
  (HISTOGRAMA)(L DE HISTO)!(ARCHIVO)(L DE AR) !
   (RUTINA)(Z TIPO)(IDENTIF)(ZONA DE PAR FOR)(INICIO)(P)!(FORMATO)(L DE FOR);
P(L DE AR)=(IDENTIF)'='(TEXT)(R ARCH);
P(R ARCH)='?'(IDENTIF)'='(TEXT)(R ARCH)!;
P(L DE CTES)=(DEF DE CTE)(RD1);
P(RD1)='?'(DEF DE CTE)(RD1)!;
P(DEF DE CTE)=(IDENTIF)'='(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)!;
```

%%%%%%%%%%% C O N D I C I O N A L %%%%%%%%%%

```
P(CUNO)=(P SI)(RC1);
P(RC1)=(SINO)(P GRAL)!;
P(P SI)=(SI)(EXP)(ENTONCES)(P INC);
      %%%%%%%%%%%%%%%%     I T E R A C I O N E S     %%%%%%%%%%%%%%%%%
P(ITER)=(REPITE)(CUERPO ITER);
P(CUERPO ITER)=(INICIO)(L ETIQ)(P GRAL)!(P INC)(HAS)(TERMINA)!
(CON)(IDENTIF)':='(EXP)(SUMANDO)(EXP)(HASTA)(EXP)(ESTO)(P INC)!
   (MIENTRAS)(EXP)(ESTO)(P INC);
P(HAS)=(HASTA)(EXP);
        %%%%%%%%%%%%%%%%   UER   %%%%%%%%%%%%%%%%%
P(P UER)=(UER)(NUMERO);
        %%%%%%%%%%%%%%%%   LLAMADO    %%%%%%%%%%%%%%%%%
P(LLAMADO)=(IDENTIF)(ZONA PAR);
P(ZONA PAR)=(ZONA DE PAR REALES)!;
        %%%%%%%%%%%%%%%%%%%%%%  A S I G N A C I O N   %%%%%%%%%%%%%%%%%
P(ASIG)=(EST REF)':=' (EXP);
P(EST REF)=(IDENTIF)(OPER ESTRUC) ;
P(OPER ESTRUC)='.'(OP EST)!(EXP SELEC);
P(OP EST)=(TITREN)!(TITCOL)!(NOMREN)!(NOMCOL)!
  (TITULO)!(MARCOCOL)!(MARCOREN)!
  (CUERPO);
      %%%%%%%%%%%%%%%%%   ENTRADA Y SALIDA   %%%%%%%%%%%%%%%%%
P(LECTURA)=(LEE)(DE)(IDENTIF)(FIN ARCH)(CON)(FORM)(L DE RECEP);
P(FIN ARCH)=(FIN) (IDENTIF)!;
P(FORM)=(LIBRE)!(L DE TEX)!(IDENTIF);
P(L DE RECEP)=(ESTO)(EST REF)(RL3)!;
P(RL3)='?'(EST REF)(RL3)!;
P(ESCRITURA)=(ESCRIBE)(EN)(IDENTIF)(CON)(FORM)(L DE ELEM A ESC);
P(L DE ELEM A ESC)=(ESTO)(EXP)(RE5)!;
P(RE5)='?'(EXP)(RE5)!;
        %%%%%%%%%%%%%%%%%%%%  S E L E C  %%%%%%%%%
P(SELEC)=(LA)(EXP)(DE)(INICIO)(L ETIQ)(P GRAL);
      %%%%%%%%%%%%%  E X P  %%%%%%%%%%%%%%%%
P(EXP)=(EXP SIMPLE) (R3);
P(R3)=(OP DE REL) (EXP SIMPLE)! ;
P(OP DE REL)='<=' ! '<>' ! '<' !'=' ! '>=' ! '>';
P(EXP SIMPLE)= (OP UNARIO) (TERMINO) (R4);
P(OP UNARIO)='+' ! '-' ! ;
P(R4)=(OP SUMA) (TERMINO)(R4) !;
P(OP SUMA)='+' ! '-'!(O);
P(TERMINO)=(FACTOR) (R5) ;
P(R5)= (OP MULT) (FACTOR) (R5) !;
P(OP MUL)='*' ! '/'!'<'!(Y);
P(FACTOR)=(OPERANDO)(R6);
P(R6)='**' (OPERANDO)!'@' !''' ! ;
P(OPERANDO)=(PRIMARIO)(EXP SELEC)!(NO)(OPERANDO);
P(PRIMARIO)=(NUMERO)!(TEXT)!(IDENTIF)(ACPU)!(PI)!(E)!
  '(' (EXP) (GRAF O REAL ARR) !
    '|' (EXP) '|'! (ARREGLO EXP);
P(GRAF O REAL ARR)=')'!'?'(EXP)'?'(EXP)')';
P(ACPU)='.'(OP EST)!(ZONA DE PAR REALES)!;
```

```
P(ARREGLO EXP)='[[' (LIMITES EXP)
    (LISTA DE EXP REP) ']]';
P(LIMITES EXP)=(NUMERO) (R7);
P(R7)= '?'(NUMERO) (R7) !;
P(LISTA DE EXP REP)=(EXP REP)(R8);
P(R8)= '?' (EXP REP)(R8)!;
P(EXP REP)= (NUMERO) '('(LISTA DE EXP REP) ')' !
     (EXP);
P(EXP SELEC)=(EXP SELEC DE ARR) !;
P(EXP SELEC DE ARR)='[' (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) (INTERVALO);
P(INTERVALO)='_' (EXP) !;
P(ZONA DE PAR REALES)='(' (LISTA DE PAR REAL) ')'
P(LISTA DE PAR REAL)=(EXP) (R14);
P(R14)='?' (EXP) (R14) ! ;
P(IDENTIF)=(IDENT);
P(TEXT)=(TEX);
P(NUMERO)=(NUM);
*
```

Por peculiaridad del BASIC de la NOVA 3/12, el símbolo "?" se utilizó en lugar del símbolo "," pues este último delimita cadenas de caracteres. El programa internamente hace la conversión.

C.2 Gramática interna

```
TYPE LJGGDIAGNOS
NUMERO DE SIMBOLOS EN LA TABLA=  798

TABLA GRAMATICAL

0       5       6       1259    0
0       9       64      12      29
13      0       0       16      351
18      369     20      519     22
36      0       0       26      401
28      356     0       0       34
794     1258    29      35      0
0       40      48      41      0
0       46      2002    23      41
47      0       0       51      421
54      2001    6       56      411
58      406     60      458     62
496     63      0       0       68
2003    117     71      2004    176
74      2005    176     73      2006
141     157     61      2028    217
84      2029    217     87      2030
238     90      2031    101     97
2014    316     786     257     2001
6       100     2013    321     0
0       107     786     1261    790
108     0       0       115     1244
786     1261    790     108     116
0       0       121     129     122
0       0       127     1244    129
122     128     0       0       134
786     1261    135     0       0
138     206     140     790     0
0       145     2004    150     147
2005    149     150     0       0
153     2024    155     2025    156
0       0       161     169     162
0       0       167     1244    163
162     166     0       0       175
176     1291    183     1293    0
0       180     786     181     0
0       186     1244    786     181
187     0       0       192     200
193     0       0       198     1244
200     193     199     0       0
205     206     1258    206     0
0       210     562     211     0
```

0	214	794	216	786
0	0	221	223	222
0	0	227	1244	229
222	228	0	0	237
176	1291	206	1244	206
1293	0	0	242	250
243	0	0	248	1244
250	243	249	0	0
256	176	1291	206	1293
0	0	262	1240	264
1241	263	0	0	269
277	176	270	0	0
275	277	176	270	276
0	0	281	2023	2004
283	284	0	0	287
302	289	2004	291	2005
293	294	0	0	297
2028	299	2029	301	2030
0	0	309	2006	141
1291	310	1293	0	0
314	1244	310	315	0
0	319	2004	320	0
0	325	333	326	0
0	331	1244	333	326
332	0	0	338	786
1261	339	0	0	343
790	344	0	0	349
1244	790	344	350	0
0	355	362	356	0
0	360	2008	13	361
0	0	368	2007	528
2009	36	0	0	373
2010	374	0	0	379
2001	29	13	383	36
401	41	395	2019	786
1258	1261	528	2017	528
2026	528	2011	36	400
2012	528	2011	36	0
0	405	2026	528	0
0	410	2039	794	0
0	415	786	416	0
0	419	768	420	0
0	427	428	1258	1261
528	0	0	432	786
433	0	0	437	1246
440	439	715	0	0
443	2032	445	2015	447
2033	449	2034	451	2035
453	2036	455	2037	457
2038	0	0	467	2016
2018	786	468	2019	474

482	0	0	472	2002
786	473	0	0	477
2021	479	339	481	786
0	0	487	2011	428
439	488	0	0	494
1244	428	489	495	0
0	504	2027	2022	786
2019	474	505	0	0
518	2011	528	512	511
0	0	517	1244	528
512	518	0	0	527
2042	528	2018	2001	29
13	0	0	532	556
533	0	0	537	539
556	538	0	0	543
1260	1261	546	1260	1262
548	1260	550	1261	553
1262	1261	555	1262	0
0	561	562	584	569
0	0	565	1243	567
1245	568	0	0	574
576	584	569	575	0
0	579	1243	581	1245
583	2020	0	0	588
607	589	0	0	594
596	607	589	595	0
0	599	1242	601	1247
604	1262	1260	606	2044
0	0	611	623	612
0	0	617	1242	1242
623	619	1264	621	1239
622	0	0	627	631
715	630	2043	623	0
0	634	794	636	790
639	786	664	641	2040
643	2041	647	1240	528
654	651	1233	528	1233
653	672	0	0	657
1241	663	1244	528	1244
528	1241	0	0	668
1246	440	670	768	671
0	0	681	1291	1291
682	1258	694	1293	1293
0	0	686	794	687
0	0	692	1244	794
687	693	0	0	698
706	699	0	0	704
1244	706	699	705	0
0	712	794	1240	694
1241	714	528	0	0
718	720	719	0	0

725	1291	726	1293	0
0	730	738	731	0
0	736	1244	738	731
737	0	0	741	1246
743	745	744	0	0
749	757	750	0	0
755	1238	757	750	756
0	0	761	528	762
0	0	766	1295	528
767	0	0	773	1240
774	1241	0	0	778
528	779	0	0	784
1244	528	779	785	0
0	789	1020	0	0
793	1040	0	0	797
1010	0	0		

C.3 Relación de símbolos no terminales

Se incluyen los de las producciones tipo B y P.

NOMBRE	TRADUCCION O LOCALIDAD EN LA GRAM. INTERNA	NUMERO CONSECUTIVO
FIN	2002	-1
CTE	2003	-2
REAL	2004	-3
TEXTO	2005	-4
ARREGLO	2006	-5
GRAFICA	2028	-6
TABLA	2029	-7
HISTOGRAMA	2030	-8
ARCHIVO	2031	-9
RUTINA	2014	-10
VAL	2023	-11
REN	2024	-12
COL	2025	-13
CUERPO	2038	-14
SINO	2008	-15
SI	2007	-16
ENTONCES	2009	-17
REPITE	2010	-18
HASTA	2026	-19
MIENTRAS	2012	-20
ESTO	2011	-21
INICIO	2001	-22
LEE	2016	-23
SUMANDO	2017	-24
DE	2018	-25
CON	2019	-26
O	2020	-27
LIBRE	2021	-28
EN	2022	-29
ESCRIBE	2027	-30
TITREN	2032	-31
NOMREN	2033	-32
NOMCOL	2034	-33

TITULO	2035	-34
MARCOCOL	2036	-35
MARCOREN	2037	-36
VER	2039	-37
LA	2042	-38
FORMATO	2013	-39
TITCOL	2015	-40
Y	2044	-41
NO	2043	-42
NUM	1010	-43
IDENT	1020	-44
TEX	1040	-45
PI	2040	-46
E	2041	-47
PRINCIPIO	1	-48
P	6	-49
DECL	64	-50
L ETIQ	29	-51
P GRAL	13	-52
COND	351	-53
ITER	369	-54
SELEC	519	-55
P INC	36	-56
RFI	23	-57
HAS	401	-58
RC1	356	-59
NUMERO	794	-60
PR INC	48	-61
TERMINA	41	-62
ASIG	421	-63
LLAMADO	411	-64
P VER	406	-65
LECTURA	458	-66
ESCRITURA	496	-67
L DE CTES	117	-68
L DE IO	176	-69
TIPO ARR	141	-70
L DE ARREGLOS	157	-71
L DE GRAF TAB	217	-72
L DE HISTO	238	-73
L DE AR	101	-74
Z TIPO	316	-75
IDENTIF	786	-76
ZONA DE PAR FOR	257	-77
L DE FOR	321	-78
TEXT	790	-79
R ARCH	108	-80
DEF DE CTE	129	-81
RD1	122	-82
DEF	135	-83
NUM O CONS	206	-84

REN COL	150	-85
DEF ARR	169	-86
RD4	162	-87
L DEF DIM	188	-88
RD5	181	-89
LIMITES	200	-90
RD55	193	-91
OP UNARIO	562	-92
N O C	211	-93
DEF GT	229	-94
RD6	222	-95
DEF HIST	250	-96
RD8	243	-97
L DE PAR FOR	264	-98
ZONA TIPO VAL	277	-99
RD11	270	-100
TIPO	284	-101
ARREG	302	-102
ESTRUCTU	294	-103
NO DIM	310	-104
DEF DE FOR	333	-105
RF1	326	-106
L DE TEX	339	-107
RLC	344	-108
P SI	362	-109
EXP	528	-110
CUERPO ITER	374	-111
ZONA PAR	416	-112
ZONA DE PAR REALES	768	-113
EST REF	428	-114
OPER ESTRUC	433	-115
OP EST	440	-116
EXP SELEC	715	-117
FIN ARCH	468	-118
FORM	474	-119
L DE RECEP	482	-120
RL3	489	-121
L DE ELEM A ESC	505	-122
RE5	512	-123
EXP SIMPLE	556	-124
RJ	533	-125
OP DE REL	539	-126
TERMINO	584	-127
R4	569	-128
OP SUMA	576	-129
FACTOR	607	-130
R5	589	-131
OP MULT	596	-132
OPERANDO	623	-133
R6	612	-134
PRIMARIO	631	-135

ACPV 664 -136
GRAF O REAL ARR 654 -137
ARREGLO EXP 672 -138
LIMITES EXP 682 -139
LISTA DE EXP REP 694 -140
R7 687 -141
EXP REP 706 -142
R8 699 -143
EXP SELEC DE ARR 720 -144
LISTA DE DIM 726 -145
ELEM DE DIM Y PTO 738 -146
R12 731 -147
ELEM DE DIM 745 -148
GRUPO DE ELEM 757 -149
R13 750 -150
INTERVALO 762 -151
LISTA DE PAR REAL 774 -152
R14 779 -153
R

C.4 Palabras Reservadas y sus localidades en Diccionario (para inicializarlo).

LOC. IVAR Localidad en Diccionario

LOC. CHARS Localidad en la Tabla de Caracteres

TYPE LJGGRELPC

PAL. CLAV.	LONG..	LOC. CHARS.	LOC. IVAR.
)INICIO	6	1	1
FIN	3	8	2
CONSTANTE	9	12	3
REAL	4	22	4
TEXTO	5	27	5
ARREGLO	7	33	6
SI	2	41	7
SINO	4	44	8
ENTONCES	8	49	9
REPITE	6	58	10
ESTO	4	65	11
MIENTRAS	8	70	12
FORMATO	7	79	13
RUTINA	6	87	14
TITCOL	6	94	15
LEE	3	101	16
SUMANDO	7	105	17
DE	2	113	18
CON	3	116	19
0	1	120	20
LIBRE	5	122	21
EN	2	128	22
VAL	3	131	23
REN	3	135	24
COL	3	139	25
HASTA	5	143	26
ESCRIBE	7	149	27
GRAFICA	7	157	28
TABLA	5	165	29
HISTOGRAMA	10	171	30
ARCHIVO	7	182	31
TITREN	6	190	32
NOMREN	6	197	33
NOMCOL	6	204	34
TITULO	6	211	35
MARCOCOL	8	218	36
MARCOREN	8	227	37
CUERPO	6	236	38
VER	3	243	39
PI	2	247	40
E	1	250	41
LA	2	252	42
NO	2	255	43
Y	1	258	44
R			

C.5 Traducción de palabras reservadas para inicializar la tabla de caracteres

Palabras Reservadas

```
TYPE LJGGPALCL7
INICIO  FIN   CONSTANTE  REAL  TEXTO
ARREGLO  SI  SINO  ENTONCES  REPITE  ESTO  MIENTRAS
FORMATO  RUTINA TITCOL  LEE  SUMANDO  DE  CON  O
 LIBRE  EN  VAL  REN  COL  HASTA
ESCRIBE  GRAFICA  TABLA  HISTOGRAMA  ARCHIVO  TITREN
  NUMREN  NUMCOL  TITULO  MARCOCOL  MARCOREN
CUERPO  VER   P)   E  LA NO Y
%
R
```

Localidad en Tabla de Caracteres

```
TYPE LJGGPCVDT
 44
 1, 8, 12, 22, 27, 33, 41, 44, 49, 58
 65, 70, 79, 87, 94, 101, 105, 113, 116, 120
 122, 128, 131, 135, 139, 143, 149, 157, 165, 171
 182, 190, 197, 204, 211, 218, 227, 236, 243, 247
 250, 252, 255, 258,
R
```

Traducción

```
TYPE LJGGPCCHAR
 259
 6, 73, 78, 73, 67, 73, 79, 3, 70, 73
 78, 9, 67, 79, 78, 83, 84, 65, 78, 84
 69, 4, 82, 69, 65, 76, 5, 84, 69, 88
 84, 79, 7, 65, 82, 82, 69, 71, 76, 79
 2, 83, 73, 4, 83, 73, 78, 79, 8, 69
 78, 84, 79, 78, 67, 69, 83, 6, 82, 69
 80, 73, 84, 69, 4, 69, 83, 84, 79, 8
 77, 73, 69, 78, 84, 82, 65, 83, 7, 70
 79, 82, 77, 65, 84, 79, 6, 82, 85, 84
 73, 78, 65, 6, 84, 73, 84, 67, 79, 76
 5, 76, 69, 69, 7, 83, 85, 77, 65, 78
 68, 79, 2, 68, 69, 3, 67, 79, 78, 1
 79, 5, 76, 73, 66, 82, 69, 2, 69, 78
 3, 86, 65, 76, 3, 82, 69, 78, 3, 67
 79, 76, 5, 72, 65, 83, 84, 65, 7, 69
 83, 67, 82, 73, 66, 69, 7, 71, 82, 65
 78, 73, 67, 65, 5, 84, 65, 66, 76, 65
 10, 72, 73, 83, 84, 79, 71, 82, 65, 77
 65, 7, 65, 82, 67, 72, 73, 86, 79, 6
 84, 73, 84, 82, 69, 78, 6, 78, 79, 77
 82, 69, 78, 6, 78, 79, 77, 67, 79, 76
 6, 84, 73, 84, 85, 76, 79, 8, 77, 65
 82, 67, 79, 67, 79, 76, 8, 77, 65, 82
 67, 79, 82, 69, 78, 6, 67, 85, 69, 82
 80, 79, 3, 86, 69, 82, 2, 80, 73, 1
 69, 2, 76, 65, 2, 78, 79, 1, 99,
R
```

APENDICE D

A continuación se presentan tres ejemplos de problemas tomados de los cursos de Estadistica proporcionados por profesores del Laboratorio de Estadistica de la Facultad de Ciencias.

EJEMPLO 1:

```
INICIO

%     SE TIENE UN CONJUNTO DE DATOS QUE REPRESENTA PESOS
MEDIDOS  EN KILOGRAMOS DE 40 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.  SE HARAN 8 INTERVALOS LOS CUALES SERAN
   CERRADOS POR LA DERECHA Y ABIERTOS POR LA IZQUIERDA.

2) HISTOGRAMA PARA LAS FRECUENCIAS SIMPLES Y ACUMULADAS

3) LOS POLIGONOS DE FRECUENCIAS SIMPLES Y ACUMULADAS  ;

CONSTANTE
     W=2,
     LIMITEINFERIOR=92,
     NUMINT=8,
     MAXDATOS=100;

REAL
     NUMDATOS;

ARREGLO
     DATOS[1:MAXDATOS],
     LIMITES,FRECSIMPLES,FRECACUM[1:NUMINT];
TABLA
     TABFREC[NUMINT,4];

HISTOGRAMA
     HFREC[NUMINT];

GRAFICA
     GFREC[NUMINT,NUMINT];

ARCHIVO
     PESOS="PESOS",
     LISTADO="LISTADO";

RUTINA LEEDATOS(ARREGLO REAL [] DATOS
                REAL NUMDATOS)
```

```
INICIO
REAL
     I,FINARCHIVO;

FORMATO
     F="F5.2";

FINARCHIVO:=0;
I:=1;
LEE EN PESOS FIN FINARCHIVO CON F ESTO A[I];
REPITE MIENTRAS NO FINARCHIVO ESTO
INICIO
     I:=I+1;
     LEE EN PESOS FIN FINARCHIVO CON F ESTO
       A[I];
FIN;
NUMDATOS:=I-1
FIN; % LEEDATOS;

RUTINA ARMAFRECUENCIAS(ARREGLO[] DATOS
            REAL NUMDATOS
            ARREGLO[] LIMITES,F,FF)
INICIO
REAL
     I,LI;

LI:=LIMITEINFERIOR;
REPITE PARA I:=1 SUMANDO 1 HASTA NUMINT ESTO
INICIO
     LIMITES[I]:=LI;
     F[I]:=((LI<=DATOS[1_NUMDATOS]) Y
           (DATOS[1_NUMDATOS]<(LI+W)))[.];
     FF[I]:=(F[1_I])[.];
     LI:=LI+W
FIN
FIN; % ARMAFRECUENCIAS;


%  SE LEEN LOS DATOS ;

LEEDATOS(DATOS,NUMDATOS);

%  SE ARMAN LAS FRECUENCIAS;

ARMAFRECUENCIAS(DATOS,NUMDATOS,LIMITES,
          FRECSIMPLES,FRECACUM);

%  SE ARMA LA TABLA DE FRECUENCIAS;

TABFREC.TITULO:="TABLA DE FRECUENCIAS";
TABFREC.NOMCOL:=[[4:"LIM INF", "LIM SUP",
  "FREC. SIMPLE", "FREC. ACUMULADA"]];
TABFREC.CUERPO[,1] := LIMITES;
```

```
TABFREC.CUERPO[,2] := LIMITES + W;
TABFREC.CUERPO[,3] := FRECSIMPLES;
TABFREC.CUERPO[,4] := FRECACUM;

%  SE ESCRIBE LA TABLA;

ESCRIBE EN LISTADO CON LIBRE ESTO TABFREC:

%  SE ARMA EL HISTOGRAMA DE FREC. SIMPLES;

HFREC.TITULO:="HISTOGRAMA DE FRECUENCIAS SIMPLES";
HFREC.CUERPO:=FRECSIMPLES;

%  SE ESCRIBE EL HISTOGRAMA DE FREC. SIMPLES;

ESCRIBE EN LISTADO CON LIBRE ESTO HFREC;

ESCRIBE EN LISTADO CON "14HFRECUENCIAS: ,",
    "20(F7.2)" ESTO FRECSIMPLES;

% SE ARMA EL POLIGONO DE FRECUENCIAS SIMPLES;

GFREC.TITULO:="POLIGONO DE FRECUENCIAS SIMPLES";
GFREC.CUERPO:=(LIMITES+0.5,FRECSIMPLES,"*");
% (SE UNEN LOS PUNTOS MEDIOS);
GFREC.TITREN:="FRECUENCIAS SIMPLES";
GFREC.TITCOL:="X";

%  SE ESCRIBE EL POLIGONO DE FREC. SIMPLES;

ESCRIBE EN LISTADO CON LIBRE ESTO GFREC;

%  SE ARMA EL HISTOGRAMA DE FRECUENCIAS ACUMULADAS;

HFREC.TITULO:= "HISTOGRAMA DE FRECUENCIAS ACUMULADAS";
HFREC.CUERPO:= FRECACUM;

%  SE ESCRIBE EL HISTOGRAMA DE FREC. ACUMULADAS;

ESCRIBE EN LISTADO CON LIBRE ESTO HFREC;

ESCRIBE EN LISTADO CON "14HFRECUENCIAS: ,",
     "20(F7.2)" ESTO FRECACUM;

%  SE ARMA EL POLIGONO DE FREC. ACUMULADAS;

GFREC.TITULO:="POLIGONO DE FRECUENCIAS ACUMULADAS";
GFREC.CUERPO:=(LIMITES,FRECACUM,"*");
% (SE UNEN LOS LIMITES SUPERIORES);
GFREC.TITREN:= "FRECUENCIAS ACUMULADAS";
GFREC.TITCOL:= "X";

%  SE ESCRIBE EL POLIGONO DE FRECUENCIAS ACUMULADAS;
```

```
ESCRIBE EN LISTADO CON LIBRE ESTO GFREC;
FIN; % PROGRAMA;
```

```
EJEMPLO 2

INICIO

%    LA  PRECIPITACION  ANUAL ES MEDIDA CADA ANO DURANTE
19 ANOS Y ESTA MEDIDA SE EXAMINA PARA VER SI LA CANTIDAD
DE  PRECIPITACION  TIENDE A CRECER O DECRECER. LO QUE SE
DESEA  PROBAR  ES LA HIPOTESIS DE INDEPENDENCIA ENTRE LA
CANTIDAD  DE PRECIPITACION Y EL ANO POR LA PRUEBA RHO DE
SPEARMAN.

HO: PRECIPITACION Y ANO SON INDEPENDIENTES.
H1: EXISTE  TENDENCIA  A  APAREAR  VALORES  GRANDES  DE
    PRECIPITACION  CON  VALORES  GRANDES  DE ANOS O VICE
    VERSA.

    SEA  ALFA  = 0.05. LOS VALORES DE W.025 Y W.975 SON
-0.4579 Y 0.4579 RESPECTIVAMENTE;

CONSTANTE
     MAXDATOS=100,
     ALFA=0.05,
     W025=-0.4579,
     W975=0.4579;

REAL
     NUMDATOS,
     VALORRHO;

ARREGLO
     PRECIPITACIONES,RANGOSP,RANGOSA,
      ANOS[1 : MAXDATOS];

ARCHIVO
     PRECIPANO="DATOS",
     LISTADO="LISTADO";

RUTINA LEEDATOS(ARREGLO[] PRECIPITACION, ANO
  REAL NUMDATOS)

INICIO

REAL
     I,FINARCHIVO;

FORMATO
     F="F5.2,I4";

FINARCHIVO:=0;
I:=1;
LEE EN PRECIPANO FIN FINARCHIVO CON F ESTO
   PRECIPITACION[I],ANO[I];
REPITE MIENTRAS NO FINARCHIUO ESTO
```

```
INICIO
      I:=I+1;
      LEE EN PRECIPANO FIN FINARCHIVO CON F
       ESTO PRECIPITACION[I], ANO[I];
FIN;
NUMDATOS:=I-1;
FIN; % LEEDATOS;

RUTINA ASIGNARANGOS(ARREGLO[] DATOS, RANGOS
      REAL NUMDATOS)

INICIO

REAL
      I;

REPITE PARA I:=1 SUMANDO 1 HASTA NUMDATOS ESTO

RANGOS[I]:=(DATOS[I]>DATOS[1_NUMDATOS])[.] +
  ((DATOS[I]=DATOS[1_NUMDATOS])[.]+1)/2;

FIN; % 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; % RHO;


%  SE LEEN LOS DATOS;

LEEDATOS(PRECIPITACIONES,ANOS,NUMDATOS);

%  SE ASIGNAN RANGOS A PRECIPITACIONES Y ANOS;

ASIGNARANGOS(PRECIPITACIONES,RANGOSP,NUMDATOS);
ASIGNARANGOS(ANOS,RANGOSA,NUMDATOS);

%  SE CALCULA EL VALOR DE RHO;

VALORRHO:=RHO(RANGOSP,RANGOSA,NUMDATOS);

%  SE HACE LA PRUEBA (DOS COLAS)
   SE RECHAZA HO SI RHO<W.025 O RHO>W.975 ;

SI  (VALORRHO<W025) O (VALORRHO>W0975) ENTONCES
       ESCRIBE EN LISTADO CON
```

```
            "13HSE RECHAZA HO"
SINO
        ESCRIBE EN LISTADO CON
          "16HNO SE RECHAZA HO";

FIN; % PROGRAMA;
```

```
EJEMPLO 3

INICIO


%     SE QUIERE AJUSTAR UN MODELO DE LA FORMA
 Y = ALFA + BETA * X + E
A UN CONJUNTO DE DATOS.

1)OBTENER LOS ESTIMADORES DE ALFA Y BETA
2)OBTENER EL COEFICIENTE DE DETERMINACION
3)OBTENER LA TABLA DE ANDEVA (ANALISIS DE VARIANZA) ;

CONSTANTE
      MAXDATOS=100;

REAL
      NUMDATOS,ALFAEST,BETAEST,R2,XMED,YMED,
      SUMACUADREG,SUMACUADTOT,SUMACUADRESID;

ARREGLO
      X1,Y1[1:MAXDATOS];

TABLA
      ANDEVA[3,4];

ARCHIVO
      PUNTOS="DATOS",
      LISTADO="LISTADO";

RUTINA LEEDATOS(ARREGLO[] X1, Y1
                REAL NUMDATOS)
INICIO
REAL
      I,FINARCHIVO;

FORMATO
      F="I3,F5.2";

FINARCHIVO:=0;
I:=1;
LEE EN PUNTOS FIN FINARCHIVO CON F ESTO X1[I],Y1[I];
REPITE MIENTRAS NO FINARCHIVO ESTO
INICIO
      I:=I+1;
      LEE EN PUNTOS FIN FINARCHIVO CON F ESTO
        X1[I],Y1[I];
FIN;
NUMDATOS:=I-1
FIN;  % LEEDATOS;


%  SE LEEN LOS DATOS;
```

```
LEEDATOS(X1,Y1,NUMDATOS);

%  SE CALCULA EL ESTIMADOR DE BETA Y ALFA;

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 ESTIMADORES DE ALFA Y BETA;

ESCRIBE EN LISTADO CON "15HALFA ESTIMADA= ,F8.5,/",
  "15HBETA ESTIMADA= ,F8.5" ESTO ALFAEST,BETAEST;

%  SE CALCULA EL COEFICIENTE DE DETERMINACION;

SUMACUADREG:=(((ALFAEST+BETAEST*X1[1_NUMDATOS]) -
   YMED)**2)[.];
SUMACUADTOT:=((Y1[1_NUMDATOS]-YMED)**2)[.];
R2:=SUMACUADREG /  SUMACUADTOT;

%  SE ESCRIBE EL COEFICIENTE DE DETERMINACION;

ESCRIBE EN LISTADO CON
  "35HEL COEFICIENTE DE DETERMINACION ES ,",
  "F8.5" ESTO R2;

%  SE ARMA LA TABLA DE ANDEVA;

ANDEVA.TITULO:= "TABLA DE ANDEVA";
ANDEVA.NOMCOL:=[[4:"GRADOS DE LIBERTAD",
  "SUMA DE CUADRADOS","CUADRADOS MEDIOS",
  "F" ]];
ANDEVA.TITREN:= "FUENTE DE VARIACION";
ANDEVA.NOMREN:= [[3:"REGRESION",
  "RESIDUAL", "TOTAL"]];
ANDEVA.CUERPO[,1]:=[[3:1,(NUMDATOS-2),
  (NUMDATOS-1)]];
SUMACUADRESID:=((Y1 - (ALFAEST +
  BETAEST*X1[1_NUMDATOS]))**2)[.];
ANDEVA.CUERPO[,2]:=[[3:SUMACUADREG,
  SUMACUADRESID,SUMACUADTOT]];
ANDEVA.CUERPO[1_2,2]:=[[2:SUMACUADREG,
  (SUMACUADRESID/(NUMDATOS-2))]];
ANDEVA.CUERPO[1,3]:=(SUMACUADREG *
  (NUMDATOS-2)) / SUMACUADRESID;

%  SE ESCRIBE LA TABLA;

ESCRIBE EN LISTADO CON LIBRE ESTO ANDEVA;
```

```
FIN; % PROGRAMA;
```

BIBLIOGRAFIA

[CH1] Chas, L.L: Statistics: Methods and Analysis. Mc Graw-Hill, 1969

[D1] Data General Corp.: Diskette Operating System Reference Manual. Data General Corp. Westboro, Mass. 1976

[D2] ---- :Extended Basic User's Manual. Data General Corp. Westboro, Mass. 1975

[D3] ---- :FORTRAN IV User's Manual. Data General Corp. Westboro, Mass. 1975

[D4] ---- :Text Editor User's Manual. Data General Corp. Westboro, Mass. 1975.

[D5] De Remer, F.: "Lexical Analysis" Compiler Construction (ed. Baier, F y Eickel, J). Springer Verlag. New York 1976

[F1] Finney, D.: An Introduction to the Theory of Experimental Design. The University of Chicago Press 1960

[G1] Graybill, F: Introduction to Matrices with Applications in Statistics. Wadsworth Publishing Company. Belmont, California 1969

[G2] Gries, D: Compiler Construction For Digital Computers. John Willey & Sons. New York 1971

[H1] Hicks, C: Fundamental Concepts in the Design of Experiments. Holt, Rinehart and Winston. New York 1964

[H2] Hoare, C.A.R.: "Record Handling" Programming Languages (ed. Genuys, F.) Academic Press 1968

[I1] IBM Corp.: APL/360 User's Manual. IBM CORP. 1968

[J1] Jensen, K y Wirth N.: Pascal User Manual and Report. Springer Verlag 1974

[K1] Kempthorne, O: The Analysis of Experiments. John Wiley & Sons. New York 1967

[L1] Lang, S: Linear Algebra. Addison Wesley Publishing Company 1971

[L2] Legarreta, L: Compiladores. Fundación Arturo

Rosenbluth. Mex. D.F. 1983

[M1] Mood,A.M y F.A.Graybill:Introduction to the Theory of Statistics. Mc Graw Hill 1963

[N1] Naur,P:"Report of the Algorithmic Language Algol 60". ACM,6,1. 1963

[N2] Nie,N.H.:SPSS Statistical Package for Social Sciences. Mc Graw Hill 1975

[R1] Rees,D.J.:Skimp. University of Edinburgh Department of Computer Science. Edinburgh Scotland 1974

[S1] Silverman,E.N.:Statistics a Common Sense Approach. Prindle, Weber & Schidt, Inc. 1973

[T1] Tanenbaum,A.S.:"A Tutorial on Algol 68". Computing Surveys 8,2. 1976