24.5

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
FACULTAD DE CIENCIAS

* UN MACRO PROCESADOR DE PROPOSITO GENERAL *

TESIS
QUE PARA OBTENER EL TITULO DE
MATEMATICO
PRESENTA
GRACIANO CRUZ ALMANZA
1983

INDICE

CAP-I INTRODUCCION

1.PROCESADORES DE TEXTO.

Dada la importancia para la humanidad de almacenar informacion (Historia, Geografia, etc.) en escritos como libros, revistas, periodicos, etc. Es conveniente tener herramientas adecuadas para facilitar la realizacion de dichos escritos. A estas herramientas se les conoce como Procesadores de Texto, es decir, permiten manipular cadenas de caracteres para modificar texto en general.

Los procesadores de texto se han desarrollado en base a un proceso historico dado por la necesidad. Con la aparicion de las Computadoras se han desarrollado herramientas como son los formateadores, editores, traductores, etc.

Este trabajo muestra una de las herramientas mas poderosas en el desarrollo de los procesadores de texto usando la Computadora, dicha herramienta es el Macro-procesador.

El macro procesador es un programa que permite sustituir una cadena de caracteres por otra cadena previamente definida, esta posibilidad da al usuario una serie de ventajas como son: La posibilidad de extender un lenguaje por introducir una nueva unidad sintactica, realizar edicion sistematica, el traslado de programas de una computadora a otra, etc.

En el presente trabajo se comentaran las conveniencias, usos y ejemplos concretos de Macro procesadores, para despues plantear y desarrollar un Macro procesador con el maximo de caracteristicas positivas.

2.CONVENIENCIA DE UN MACRO-PROCESADOR.

En el trabajo en una computadora, al programar se encuentra que en algunos programas existen bloques de instrucciones que se repiten varias veces, lo cual es un trabajo tedioso para los programadores. Dichos bloques pueden tener liseras deferencias entre si, estas diferencias pueden ser desde algun numero en alguna instruccion, hasta un conjunto de instrucciones.

La similitud entre los bloques de instrucciones hace conveniente declarar dicho bloque una sola vez, como un patron con argumentos que varian en las diferentes instancias del uso de dicho patron.

A las instrucciones que causan la generacion de otras instrucciones se les conoce como Macro-instrucciones o simplemente Macros.

En terminos muy generales se podria decir que un Macro da la facilidad de reemplazar una secuencia de simbolos por otra.

La estructura para la definicion de una Macro instruccion es:

```
MACRO NOMBRE( LISTA DE PARAMETROS )
{ cuerpo }
ENDM
```

En donde MACRO y ENDM son generalmente palabras reservadas que determinan el inicio y el fin de una Macro-instruccion respectivamente. A este proceso tambien se le da el nombre de Macro-definicion.

La cadena NOMBRE es un identificador asignado al Macro de tal manera que el usuario pueda definir varios Macros identificandolos con nombres diferentes.

El cuerpo es el bloque de instrucciones que determinan el patron en si.

La LISTA DE PARAMETROS son los llamados parametros formales que seran substituidos dentro el cuerpo del Macro por los parametros actuales, que seran dados por el usuario a la hora de hacer uso del Macro. Al hecho de hacer uso de un Macro se le llama Macro-llamada o Macro-expansion.

La forma comun de hacer un llamado a un Macro es:

```
NOMBRE( LISTA DE PARAMETROS ACTUALES )
```

Es decir se da el nombre del Macro al cual se quiere uno referir, asi como la lista de parametros

actuales que seran los que sustituyan en el cuerpo del Macro a los parametros formales.

3.EJEMPLOS ESPECIFICOS DEL USO DE MACRO-PROCESADORES.

Para clarificar mas la conveniencia y mostrar aplicaciones de Macros, se veran algunos ejemplos:

a) Suponsase que se tiene el siguiente programa en PASCAL:

```
BEGIN
-----
FOR I:=1 TO 10 DO X:=X+1;
--
--
FOR I:=2 TO 5 DO Y:=Y+2
--
FOR J:=10 TO 100 DO Z:=Y+Z;
--
END.
```

Notese que en este programa existe un bloque (de una instruccion) que se repite en varias ocasiones, con algunas diferencias.

En estos casos es muy conveniente poder tener un Macro, de tal manera que ayude a la programacion, por ejemplo:

```
MACRO REP(A,B,C,D)
FOR A:=B TO C DO D;
ENDM
```

Es decir un Macro de nombre REP con los parametros A,B,C,D. De tal manera que solamente sea necesario llamarlo por su nombre, y asi el programa anterior quedaria:

```
BEGIN
--
REP(I,1,10,<X:=X+1>)
--
--
REP(I,2,5,<Y:=Y+2>)
--
REP(J,10,100,<Z:=Y+Z>)
--
END.
```

Hay que hacer notar que en el programa resultante la escritura del programa es mucho mas sencilla y mas entendible.

b) Suponsase que se tiene el siguiente programa en ensamblador:

```
INICIO:ADD     12,R0
       -------
       -------
       ROL     R2
       ROL     R2
       ROL     R2
       -------
       -------
       ROL     R2
       ROL     R2
       ROL     R2
       -------
       -------
       ROL     R3
       ROL     R3
       ROL     R3
       -------
       -------
       .END INICIO
```

Si se pudiera declarar un patron (MACRO) de tal manera que su cuerpo fueran las tres instrucciones del bloque se repite varias veces(3), podria simplificarse el problema. El macro podia ser:

```
       MACRO ROTALEF(X)
       ROL     X
       ROL     X
       ROL     X
       ENDM
```

De tal manera que el programa quedaria:

```
INICIO:       ADD      12,R0
       -------
       -------
       ROTALEF(R2)
       -------
       -------
       ROTALEF(R2)
       -------
       -------
       ROTALEF(R3)
       -------
       -------
       .END    INICIO
```

4.USOS INTERESANTES DE MACROS.

a) Una funcion importante que juega el poder definir Macros es el hecho de que extiende el repertorio de instrucciones de un lenguaje, esto lo hace mas poderoso en la medida que los usuarios pueden definir nuevas instrucciones.

Por ejemplo suponsase que en FORTRAN se desea un Macro que realice la instruccion:

REPEAT <STATMENT> UNTIL <EXP. BOOLEANA>

Este proceso estara escrito con instrucciones FORTRAN y permitira el uso de la instruccion REPEAT en FORTRAN.

Esta posibilidad de construir instrucciones propias puede ser tan importante y poderosa que un usuario podria dotar a FORTRAN con herramientas tan poderosas como el hacer que FORTRAN tenga una estructura de bloque, trabajos en este sentido han sido ya hechos, como es el caso de FOREST (FORTRAN ESTRUCTURADO). Para mayor informacion consultar:[LEVINE 1].

b)El traslado de software de una computadora a otra es una de las principales aplicaciones de los Macros, por ejemplo, suponsase que se tiene un paquete estadistico en FORTRAN cargado en una maquina M1 y se desea trasladarlo a una maquina M2, pero que dicho FORTRAN difiere en algunos casos con respecto al FORTRAN de la maquina M1.

Dado este problema, se puede resolver teniendo un Macro-expansor en la computadora M2 de tal manera que dicho Macro-expansor sea el encargado de realizar la "traduccion" del FORTRAN de la maquina M1 al FORTRAN de la maquina M2. En donde por "traducir" se entendera substituir las proposiciones del FORTRAN en M1 que no son admisibles para el compilador FORTRAN de la maquina

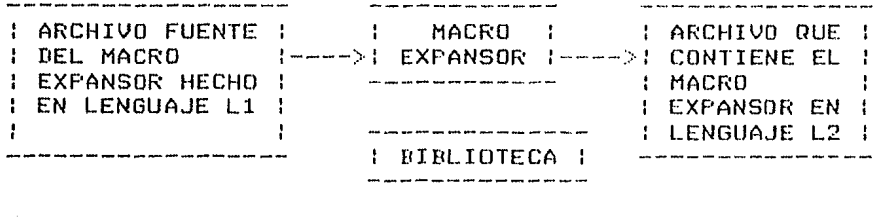
M2 por proposiciones que si lo sean. Dicha substitucion de proposiciones se hara por sus equivalentes al FORTRAN de M2 o se hara por un conjunto de proposiciones en el FORTRAN de M2 que efectuen la misma funcion que la proposicion FORTRAN de M1.

Mas especificamente, en M2 se construirian un conjunto de Macros de tal manera que las proposiciones del FORTRAN de M1 que se deseen cambiar sean las llamadas a estos Macros y las expansiones de dichos Macros sean las proposiciones nuevas del FORTRAN de la maquina M2, teniendo en cuenta que las proposiciones que son comunes a los compiladores seran dejadas tal cual se encuentren.

El traslado de software es en general muy utilizado en cualquier tipo de programacion, ya sea en paquetes o programas de Sistemas Operativos.

Por ejemplo si en la maquina M1 se diseno un lenguaje L1 utilizando Macros y se desea pasarlo a la maquina M2 entonces se utiliza el Macro-expansor con los Macros respectivos.

Mas aun, suponsase que el Macro-expansor se desarrolle en M1, esto llevaria a un traslado total, aun del Macro-expansor.



Esto se realizaria dando como archivo fuente al propio Macro-expansor, y asi trasladar(transportar) el Macro-expansor de la maquina M1 a la maquina M2.

c)Otra aplicacion interesante es la llamada EDICION SISTEMATICA, lo cual se refiere a reemplazar todas las ocurrencias de un identificador dentro de un programa fuente, por otro identificador distinto.

La edicion sistematica es utilizada cuando se desarrolla software por varias personas y donde cada una de las personas realiza un modulo de software. Sucede que al reunir todos los modulos existen identificadores que son repetidos o muy semejantes. Esto puede llevar a posibles malas interpretaciones al revisar todo el conjunto de modulos y tenderia a

ocasionar errores graves y muy dificiles de detectar.

De la misma aplicacion podria pensarse en el uso de constantes dentro del programa, a manera de ejemplo definiremos el siguiente Macro:

```
MACRO LONG
15
ENDM
```

De tal manera que en el programa fuente aparezca el identificador LONG, por ejemplo:

```
DIMENSION       DATOS(LONG)
-------------------------------
-------------------------------
-------------------------------
DO 100 I=1, LONG
-------------------------------
-------------------------------
-------------------------------
END.
```

De esta forma al efectuarse las Macro-expansiones se efectue la sustitucion y el programa quedaria.

```
DIMENSION       DATOS(15)
-------------------------------
-------------------------------
-------------------------------
DO 100 I=1, 15
-------------------------------
-------------------------------
-------------------------------
END.
```

En general se puede decir que a mayor nivel del lenguaje de programacion se necesita menos de los Macros, esto es debido a que dichos lenguajes van teniendo nuevas estructuras y nuevas proposiciones de tal manera que tienden a facilitar al programador su trabajo.

Sin embargo, aun en los lenguajes de alto nivel mas comprensibles y completos, los Macros son utilizados para introducir proposiciones especialmente disenadas para la aplicacion particular del usuario.

Podemos ver el lenguaje FORTRAN, el cual es muy usado en Analisis Numerico, y es muy conveniente tener

un conjunto de proposiciones de las cuales mostramos un solo ejemplo:

```
MULT(A,B,C,I,J,L)
```

Este Macro consiste de las instrucciones en FORTRAN que realizaran la multiplicacion de las matrices A(I,J) y B(J,L); dejando el resultado en C(I,L).

CAP-II COMPARACION DE MACRO-PROCESADORES CLASICOS.

1.INTRODUCCION.

A continuacion se dara una breve descripcion de los Macro-procesadores PL/I [Nichols-1], GPM [Strachey-1], TRAC [Mooers-1] y ML/I [Brown-1]. De esta forma se podran observar sus caracteristicas principales.

Para finalizar se hara una comparacion entre MPPG (Macro-procesador de Proposito General) y de otros Macro-procesadores de tal manera que se vean las ventajas de MPPG asi como sus desventajas.

2.MACRO-PROCESADOR PL/I.

La idea basica de este procesador es que tanto la sintaxis como la semantica de las proposiciones al Macro-procesador sean las mismas que las de las proposiciones de PL/I.

PL/I utiliza un simbolo(%) de marca especial, lo cual permite que las definiciones de Macros no esten en un campo fijo (como en los Macro-ensambladores). A dicho simbolo se le conoce como caracter de senal o advertencia.

En el Macro-procesador PL/I el reemplazamiento ocurre en cualquier lugar, excepto dentro de comentarios y cadenas constantes. Esto es porque el texto no es tratado en la base de carcter por caracter.

PL/I tambien tiene requerimientos al Macro-procesador como ACTIVE y DESACTIVE .
Estas funciones permiten al usuario que parte de su texto fuente no sea examinado por el Macro-procesador, haciendo que el tiempo de procesado se reduzca (ver:[Nicholls-1]).

Este Macro-procesador tiene la facilidad de accesar bibliotecas de Macros.

La forma de llamar Macros es:

NOMBRE(ARG1,ARG2,...,ARGn).

Notese que nombre no tiene caracter especial de senal o aviso.

El Macro-procesador PL/I es realizado como un preprocesador y puede ser visto como un modulo de software separado del compilador PL/I.

Los nombres de los Macros en PL/I deben ser identificadores del tipo del lenguaje base.

Para un estudio mas detallado ver:[Nicholl-1], [Brown-3].

3.MACRO PROCESADOR GPM (General Purpose Macrogenerator).

GPM trabaja con un caracter de advertencia( ), el cual es necesario poner en las definiciones de Macros asi como en las Macro-llamadas (Macro-expansiones).

GPM es de proposito general, es decir, manipula el texto sin importar el lenguaje en el que este dicho texto.

Los Macros en GPM deben haber sido definidos previamente antes de hacer algun llamado a ellos.

GPM utiliza el caracter '~' dentro del cuerpo de los Macros para identificar donde va el parametro actual correspondiente al llamado.

GPM permite que Macro-llamadas sean parametros actuales de otras Macro-llamadas.

En GPM se utilizan los caracteres '<' y '>' para incluir como parametro actual cualquier cadena de caracteres.

En GPM al definir un Macro, el usuario mismo debe colocar ~1, ~2,...., ~n, para saber en donde iran los parametros actuales. Estas marcas evidentemente deberan estar dentro del Macro.

Si el nombre de un Macro ya habia sido definido entonces el Macro que fue definido ultimamente sera el que se expandera al hacerse un Macro-llamado.

En GPM es permitida la anidacion de Macros, es decir, se pueden definir Macros dentro de Macro-definiciones.

Una caracteristica de GPM es que cuando encuentra una Macro-llamada a un Macro, examina la llamada antes de hacer cualquier cosa. Esto permite que en el llamado del Macro pueda ir una Macro-definicion como parametro actual.

Por ejemplo:

```
A,X,U, DEF,A,<~1~2~1>;;
```

Donde se observa lo siguiente:

a)La definicion de un Macro se hace de la siguiente forma:

```
DEF,A,<~1~2~1>;
```

La funcion DEF define un Macro de nombre A y con cuerpo ~1~2~1. El cuerpo del Macro consistira de colocar el primer parametro, despues el segundo y por ultimo se coloca nuevamente el primer parametro actual.

b)El llamado al Macro A se hace de la siguiente forma:

A,X,U, DEF,A,<^1^2^1>;;

Donde se pude ver el caracter de advertencia( ), el nombre del Macro que se desea expander(A), el primer parametro actual(X), el segundo parametro actual(U) y para terminar un tercer parametro (DEF,A,<^1^2^1>;) el cual consiste en la definicion del Macro A. Esto es valido ya que antes de realizar la expansion del Macro verifica todo el texto de la Macro-llamada y como en el texto de la Macro-llamada se encuentra una Macro-definicion, entonces hace primero la Macro-definicion y despues procede a realizar la Macro-llamada. La cual produce: XUX .

Note que en este ejemplo un Macro lleva como parametro actual su propia definicion, permitiendo mayor flexibilidad.

GPM esta limitado por utilizar los simbolos ' ',';',' <' y '>', ya que los utiliza como caracteres de advertencia, sin embargo los simbolos utilizados tienen la caracteristica de que no son parte del conjunto de caracteres del lenguaje Ensamblador de la Computadora TITAN (ATLAS 2).

GPM tiene expresiones condicionales y admite definiciones que contengan Macro-llamadas. Especialmente se puede mencionar recursion, o sea, el hecho de que un Macro pueda llamarse a si mismo, esto evidentemente se detiene para que no sea un proceso infinito por las expansiones condicionales.

GPM esta escrito en Ensamblador, lo cual da la facilidad de optimizar codigo, pero tiene la caracteristica poco deseable de no ser muy claro en su lectura para un estudio detallado del Macro-procesador.

En el procesamiento del texto GPM maneja el texto fuente como una cadena de caracteres.

GPM fue desarrollado explicitamente como un preprocesador y reemplaza las Macro-llamadas en el programa para que este entre a compilarse normalmente.

Probablemente el uso mas extenso de GPM haya sido la creacion de un conjunto de Macros, basados en lenguaje Ensamblador, para el desarrollo de Programacion de Sistemas.

Para un estudio mas detallado ver:[Strachey-1], [Gries-1].

4.MACRO-PROCESADOR TRAC.

TRAC (Text Reckoning And Compiling) es similar a GPM ([Strachey-1]), pero no utiliza la misma notacion. Este Macro-procesador trabaja usando un caracter de senal( ).

TRAC es mas utilizado que GPM ya que ha tenido mas propaganda, ademas, contiene un conjunto de funciones ya construidas, las cuales son llamadas de la siguiente forma:

(NOMBRE DE LA FUNCION,CADENA1,CADENA2,...,CADENAn)

Una de las funciones es 'DS' (Define Cadena) y esta define un Macro. La forma de hacerlo es:

(DS,NOMBRE,<CUERPO DEL MACRO>)

Donde se puede observar el nombre de la funcion(DS), el nombre del Macro(NOMBRE) y el cuerpo.

Para realizar una Macro-llamada se escribe:

(CL,NOMBRE,ARG1,ARG2,...,ARGn)

CL es una funcion ya construida que llama a un Macro definido con la funcion DS.

En TRAC llamar a un Macro es un ejemplo particular de la funcion ya construida, mientras que en GPM las funciones ya construidas (el Macro DEF, por ejemplo) son ejemplos particulares de Macro-llamadas.

Con respecto a que TRAC contiene un conjunto de funciones ya definidas podremos observar que dichas funciones no son llamadas a Macros, sino que dichas funciones son parte de TRAC.

TRAC acepta tambien archivos de Macros de tal manera que el trabajo en el archivo fuente se pueda reducir a simples llamadas.

Algunos ejemplos de funciones ya definidas son:

(DS,...)_Define Macro.

(SS,...)_Lista de parametros formales.

(CL,...)_Macro-llamadas.

(ML,x,y)_x * y.

(AD,x,y)_x + y.

(EQ,x,y,s1,s2)_Si x=y entonces s1 si no s2.

En TRAC los argumentos son insertados dentro del texto de reemplazamiento como la primera accion de una Macro-llamada y entonces una decision es hecha con respecto al tratamiento del texto.

TRAC permite anidacion en llamadas, es decir, Macro-llamadas dentro de Macros y ademas admite recursividad, lo cual lo hace un lenguaje poderoso y muy utilizado.

Otra caracteristica muy particular y sobre todo poderosa de TRAC es la funcion READ, la cual lee una cadena de caracteres de algun dispositivo.

Como cadena de entrada podria pensarse en una cadena de caracteres de longitud arbitraria, dada desde una consola del usuario.

Dicha cadena seria terminada por un caracter especial(de senal), de esta forma el texto de entrada estara bajo control total del usuario y el proceso podria ser conversacional. A este proceso se le conoce como entrada controlada.

En especial podemos pensar que la cadena de entrada sea una Macro-llamada.

La forma de usar la funcion es:

(READ)

El procesamiento de esta funcion consta de interrrumpir el procesamiento sobre el archivo fuente y pasar a procesar el texto proveniente de la terminal, el cual sera dado por el usuario directamente.

Si por ejemplo el usuario da por la terminal:

(ML,A,B)

TRAC procesa la linea y como es un llamado a una Macro, pues entonces reliza la funcion 'ML'. El final de la cadena es dado por un caracter especial previamente definido.

En la mayoria de los macro-procesadores los flujos de entrada son tratados de una manera uniforme, es decir, la entrada es obtenida de un archivo con la informacion invariante. A esta forma de trabajar se le conoce como entrada no controlada.

La entrada no controlada es la forma comun del uso de Macros, es decir, el texto de entrada son proposiciones explicitamente especificadas en un archivo que sera analizado de una manera uniforme y en donde se realizaran las expansiones.

Para la entrada no controlada el Macro debe haber estado ya fisicamente en el archivo para ser tomado en cuenta.

En la entrada controlada el usuario mantiene una conversacion interactiva con TRAC de tal manera que se puede dar como cadena de entrada el propio cuerpo de un Macro. De esta forma se pueden hacer pruebas con los Macros dados desde la terminal sin tener que modificar el texto fuente.

Para un estudio mas detallado ver :[Moorse-1], [Brown-3].

5.MACRO-PROCESADOR ML/I.

ML/I es un Macro-procesador de proposito general y es mas similar a GPM que a TRAC.

ML/I fue disenado como herramienta para permitir a los usuarios extender cualquier lenguaje de programacion, al incorporar nuevas proposiciones y otras formas sintacticas de nuestra propia eleccion y en nuestra propia notacion. Esto permite un lenguaje completamente orientado al usuario.

Un macro es basicamente un medio de extender un lenguaje de programacion existente, llamado el lenguaje base, al permitir introducir una nueva unidad sintactica, la cual es descrita en terminos de unidades sintacticas existentes de el lenguaje base. ML/I actua como un preprocesador al compilador para el lenguaje base.

Varios Macro-procesadores con propiedades mas generales que los basicos Macro-ensambladores, han sido producidos, entre ellos GPM([BROWN-3,[GRIES-1]), TRAC([MOOERS-1],[BROWN-3]), LIMP([BROWN-3]) y PL/I([NICHOLLS-1]). Todos ellos han sido desarrollados en dos principales caminos.

a)El usuario puede definir su propia notacion para realizar Macro-llamadas en lenguajes especificos.

b)El Macro-procesador es independiente del lenguaje base. En este caso el mismo Macro-procesador puede actuar como un preprocesador para cualquier lenguaje.

ML/I incorpora ambas extensiones y ademas permite al usuario extender cualquier lenguaje, utilizando su propia notacion.

En ML/I los Macros pueden ser usados para hacer cambios en programas que ya hayan sido escritos. Este uso de Macros para reemplazar un patron de simbolos por otro a traves de un programa es llamado edicion sistematica y no debe confundirse con la modificacion de un programa por medio de un editor de textos.

La caracteristica principal de ML/I es que el usuario especifica una estructura de delimitadores por cada Macro, lo que hace posible que para cada uno de los Macros se tenga cualquier numero de alternativas de delimitadores.

El proposito de la estructura de delimitadores es definir el nombre o nombres del Macro y definir por cada delimitador un sucesor o conjunto de sucesores alternativos. Donde un sucesor es el siguiente delimitador a ser buscado cuando se examina un llamado.

Un terminador puede ser considerado a tener un sucesor nulo.

La estructura de delimitadores puede ser disenada para permitir Macros con una lista de longitud indefinida de argumentos.

ML/I permite Macro-llamadas anidadas dentro de argumentos de otras Macro-llamadas y tambien tiene recursividad.

ML/I es independiente del lenguaje base y podra ser usado como un preprocesador comun a todos los Compiladores de una instalacion, en el mismo camino de que cada Compilador y Ensamblador podrian usar un cargador comun.

El uso de ML/I no esta confinado a extension de lenguaje o a edicion sistematica. Otro uso es la transportacion de software, por ejemplo, el pasar software en FORTRAN IV a otra maquina que tenga FORTRAN II.

ML/I trata el texto de entrada como una secuencia de atomos en lugar de tomarlo como una cadena de caracteres individuales.

Cada vez que un nombre de un Macro es encontrado durante el examen del texto es tomado por ML/I como el inicio de una Macro-llamada y una busqueda es hecha para los delimitadores restantes.

ML/I tambien tiene la posibilidad de declarar los caracteres que determinan los comentarios, de tal manera que los textos que sean parte de los comentarios del programa sean pasados a la salida sin ser examinados por ML/I. Esto permite mayor rapidez en el procesado y aun sigue siendo independiente del lenguaje ya que el usuario declara los caracteres que son utilizados en ese lenguaje para determinar sus comentarios.

ML/I trabaja normalmente en modo libre pero puede ser colocado a modo de advertencia, es decir, usar un caracter de aviso.

Si ML/I esta en modo de advertencia, todas las Macro-llamadas deberan ser precedidas por el caracter de senal y todos los Macro-nombres que no sean precedidos por la marca de advertencia seran tratados literalmente.

Si ML/I no esta en modo de advertencia, los Macro-nombres son escencialmente palabras reservadas.

ML/I contiene un numero de Macros ya construidos llamados operaciones Macro y realizan funciones bien especificas, como por ejemplo declarar(definir) macros. los nombres de las operaciones macro principian con las letras 'MC', de tal forma que difieran de los Macros definidos por el usuario.

Por ejemplo para definir un Macro se usa la siguiente operacion macro:

MCDEF ARG1 AS ARG2.

Existen otras palabras reservadas, como: WITH, OPT, OR, ALL y cualquier atomo consistente de la letra 'N' seguida por un entero. Los nombres de palabras reservadas pueden ser cambiados dinamicamente.

Para un estudio mas detallado ver:[BROWN-1], [BROWN-2].

6.COMPARACION CON MPPG (Macro-Procesador de Proposito General).

MPPG al igual que la mayoria de los Macro-Procesadores es de Proposito General, es decir, no depende del lenguaje de Programacion utilizado (lenguaje base).

De los macro-procesadores anteriormente descritos, PL/I es el unico que es de Proposito especifico, restringiendo mucho su aplicacion.

En general la filosofia de los Macro-Procesadores es trabajar como Preprocesadores, lo cual quiere decir que son modulos independientes de los Compiladores y que el texto antes de entrar al Compilador debe haber pasado por el Macro-Procesador.

MPPG fue pensado para que trabajara como Preprocesador al igual que todos los macro-procesadores descritos anteriormente.

Otra caracteristica que tienen todos los macro-procesadores (incluyendo a MPPG) es el permitir al usuario dar su propia biblioteca de Macros, la cual estara almacenada en algun dispositivo de memoria secundaria.

MPPG trabaja en modo de advertencia al igual que PL/I, GPM y TRAC. ML/I trabaja en modo libre.

Se escogio el modo de advertencia para MPPG por su rapidez de proceso. En modo de advertencia el macro-procesador va buscando al caracter de senal y cuando lo encuentra examina los caracteres posteriores para ver si es una Macro-definicion o una macro-llamada, pero mientras no encuentre el caracter de senal, pasa el texto a la salida sin efectuar ningun proceso. En cambio el modo libre toma un identificador y va a la tabla de nombres de macros para verificar si es una macro-llamada, pero todo esto se lleva mucho tiempo de procesado ya que es necesario verificar todos y cada uno de los identificadores.

MPPG permite que se puedan cambiar los caracteres de senal dinamicamente, esta caracteristica solo la tiene ML/I.

MPPG (al igual que GPM y TRAC) no permite que el usuario defina sus caracteres de comentarios, ya que esto complica la forma de tratar el texto, debido a que cada lenguaje de programacion tiene sus propias reglas sobre los comentarios. Por ejemplo en FORTRAN un comentario es toda linea que tiene una letra 'C' en la primer columna, PASCAL utiliza '(*' y '*)' para delimitar el principio y final de comentarios respectivamente. La mayoria de los Ensambladores inician los comentarios con el caracter ';' y los terminan con el fin de linea. Asi todos los lenguajes tienen sus propias caracteristicas respecto a los

comentarios. Esto es una pequena desventaja de MPPG ya que los usuarios deberan tener cuidado de que dentro de los comentarios no aparezcan Macro-llamadas o macro-definiciones.

Una caracteristica poderosa que tienen la mayoria de los Macro-procesadores es la posibilidad de definir Macros dentro de Macro-definiciones. Esta carcateristica tambien la tiene GPM, TRAC y ML/I y evidentemente MPPG.

De todas las caracteristicas, es claro que las mas importantes son: la posibilidad de anidar Macros y la recursion. El diseno de MPPG admite anidacion y recursividad, ademas la expansion condicional realizada en base a funciones.

Un punto importante de MPPG es la realizacion del proceso interactivo, el cual unicamente es permitido por TRAC.

Resumiendo, MPPG fue disenado pensando en las caracteristicas fundamentales deseables en un Macro-procesador.

Ahora bien, MPPG no es un Macro-procesador mas, ya que basados en las caracteristicas de otros Macro-procesadores podemos indicar que MPPG tiene ventajas con respecto a los otros Macro-procesadores.

Para respaldar esta afirmacion veremos la siguiente tabla:

	PL/I	GPM	TRAC	ML/I	MPPG
PROPOSITO GENERAL	NO	SI	SI	SI	SI
FORMA DE USO	PREPRO.	PREPRO.	PREPRO.	PREPRO.	PREPRO.
BIBLIOTECA DE MACROS	SI	SI	SI	SI	SI
MODO DE TRABAJO	SENAL	SENAL	SENAL	LIBRE	SENAL
CAMBIO DE CARACTER	NO	NO	NO	SI	SI
SALTA LOS COMENTARIOS	SI	NO	NO	SI	NO
MANEJO DEL TEXTO.	ATOMO	CARACT.		ATOMO	ATOMO
ANIDACION EN MACRO DEFINICIONES.		SI	SI		SI
ANIDACION EN MACRO LLAMADAS		SI	SI	SI	SI
RECURSION		SI	SI	SI	SI
ESPANSION CONDICIONAL		SI	SI	SI	SI
FUNCIONES PREDEFINIDAS	SI	NO	SI	NO	SI
ENT/SAL. CONTROLADA	NO	NO	SI	NO	SI

En la tabla anterior se puede verificar que MPPG ofrece las mejores caracteristicas. Como se podra observar algunos Macro-procesadores caen por su propio peso, como es el caso de PL/I, el cual, aparte de ser de proposito particular tiene otras caracteristicas que los restringen mucho.

En general la mayoria de los Macro-procesadores tienen algun punto vulnerable ya que en un principio fueron construidos pensando en problemas especificos que se suscitaron en el desarrollo de otros proyectos.

Despues aunque se desarrollaron plenamente como proyectos de investigacion, aun se tenia en mente cierto uso especifico basado en las necesidades emanadas de las experiencias en esos momentos. Asi se construyeron Macro-procesadores muy poderosos y versatiles, como es el caso de GPM y TRAC.

Los ultimos avances han sido en la direccion de ML/I, el cual ha sido uno de los mas poderosos Macro-procesadores construidos, sin embargo se puede ver que son necesarias algunas caracteristicas mas, como es el caso del proceso interactivo o proposiciones de asignacion directa. Posteriormente se argumentara la importancia de estas caracteristicas ya que resultan en dar al usuario mas poder sobre el manejo del texto.

Si efectuamos una comparacion mas comentada podremos observar como GPM y TRAC son muy similares, sin embargo, el que los dos dependan de un caracter de senal (advertencia) fijo los restringe en la medida de que dependen exclusivamente de un conjunto de caracteres fijo. Esto quiere decir que si se utiliza en un lenguaje de programacion en cuyo conjunto de caracteres esta el caracter de senal entonces habra problemas graves. En MPPG no existe este problema ya que permite el cambio del caracter de senal, este cambio puede hacerse directamente al preprocesador MPPG (permanentemente) por medio de un Editor de textos o dinamicamente, es decir, al momento de llamar a MPPG existe la opcion para el cambio del caracter de advertencia.

La opcion antes descrita de MPPG aunada a la caracteristica de manejar las cadenas de entrada como atomos, son las caracteristicas que hacen a MPPG mejor que GPM y TRAC.

De hecho el Macro-procesador mas poderoso de los vistos es ML/I ya que cuenta con un gran numero de caracteristicas favorables para los usuarios (como se podra ver en la tabla) y mas aun, se pueden definir los caracteres de comentarios. Sin embargo aunque en MPPG no se pueden definir los caracteres de comentarios (las razoes fueron expuestas anteriormente), esto no constituye una desventaja definitiva ya que MPPG admite proceso interactivo y ML/I no.

Otro punto importante de MPPG es que tiene la caracteristica de admitir proposiciones de asignacion directa a un simbolo, y es de la forma:

SIMBOLO=EXPRESION.

Esta caracteristica le da a MPPG una ventaja sobre los otros Macro-procesadores, ya que en dicha caracteristica el usuario podra igualar un valor que proviene de una expresion aritmetica a un simbolo y no solo eso, sino que en la expresion podran aparecer otros simbolos que ya hayan sido definidos previamente por el usuario. Todo esto se explicara mas ampliamente en el siguiente capitulo.

MPPG tiene ventajas sobre los otros Macro-procesadores y en algunos casos considerables, sin embargo el diseno claramente no es todo lo deseable y como todo software tendra que pasar por el proceso de depuracion.

CAP-III.DESCRIPCION DE MPPG.

1.PREPROCESADOR DE PROPOSITO GENERAL.

MPPG trabaja en la base de ser un preprocesador, es decir, sigue la linea de la mayoria de los Macro-procesadores. Esto es debido a que MPPG fue pensado como un Macro-procesador de proposito general, o sea, que no dependa del lenguaje base(lenguaje de programacion utilizado).

MPPG es un modulo software separado de cualquier traductor y el texto antes de entrar a los traductores debe haber pasado por el Macro-procesador MPPG, por esto es el adjetivo de preprocesador. El siguiente dibujo muestra el proceso:

En el caso de que la aplicacion de MPPG no sea la antes expuesta, como en el uso del manejo de texto para correspondencia, entonces, el preprocesado sera el unico proceso utilizado. Evidentemente las aplicaciones seran directamente en funcion del usuario.

Existen tambien Macro-procesadores de proposito especifico, como son los Macro-ensambladores, los cuales unicamente sirven para el lenguaje Ensamblador especifico.

2.BIBLIOTECA DE MACROS.

Una caracteristica de mucha importancia debido al poder que implica es la posibilidad de tener bibliotecas de Macros.

Una biblioteca de Macros es un archivo almacenado en memoria secundaria que contiene una serie de Macro definiciones para determinada aplicacion.

Esta biblioteca contiene un conjunto de Macros que seran utilizados en varios programas o aplicaciones, y da la facilidad al usuario de no tener que declarar este conjunto de Macros en todos y cada uno de sus programas ya que resultaria un trabajo tedioso cansado y redundante. Es por esto que en un archivo se tiene una sola copia de las Macro-definiciones y se conjuntan los archivos de la siguiente forma:

Aunque fisicamente son dos archivos distintos, los podemos ver como uno solo, esto permite que los usuarios tengan el texto fuente en su archivo y no haya texto redundante con el el consiguiente desperdicio de memoria. El proceso quedaria graficamente:

Cuando se dice bibliotecas de Macros es porque se pueden tener varios archivos con Macro-definiciones de tal forma que se pueda determinar cual de todos esos archivos se va a utilizar.

MPPG admite bibliotecas de Macros, es decir, en el momento de llamar a MPPG, este pregunta por la terminal del usuario si es que existe alguna biblioteca o bibliotecas que el usuario desee utilizar. Esta pregunta se efectua de tal forma que el usuario teclee por la terminal el nombre del archivo y entonces MPPG abre el archivo, lo recorre definiendo los Macros que ahi se encuentren y colocando los nombres de dichos Macros en el directorio de Macros. El proceso de como se hace esto, asi como una breve argumentacion del porque se hara en el capitulo de realizacion.

```
>
>; EJEMPLO DEL USO DE BIBLIOTECAS.
>
>PIP TI:=BIBLIO.TXT
%MACRO M0( )
A1A   A2A    A3A
%ENDM
%MACRO M1( )
B1B   B2B    B3B
%ENDM
%MACRO M2( )
C1C   C2C    C3C
%ENDM
>
>
>
>PIP TI:=MAIN.TXT
HOLA
%M0( )
HOLA TODOS
%M1( )
HOLA TODOS LO HUMANOS
%M2( )
ADIOS TODOS LOS HUMANOS QUE ME ESCUCHARON
>
>; SE LLAMA A MPPG
>
>@A
>PIP *.TMP;*/DE
>RUN MPPG
CAMBIO DE CARACTER DE ADVERTENCIA ?
N
USARA SU PROPIA BIBLIOTECA?
Y
DEME EL NOMBRE DE SU BIBLIOTECA
BIBLIO.TXT
DEME EL NOMBRE DEL ARCHIVO FUENTE
MAIN.TXT
EL DIRECTORIO DE MACROS ES:
EL MACRO SE LLAMA M0              Y SU AP. AL ARCHIVO ES:       1
EL MACRO SE LLAMA M1              Y SU AP. AL ARCHIVO ES:       3
EL MACRO SE LLAMA M2              Y SU AP. AL ARCHIVO ES:       5
TABLA DE SIMBOLOS DEF.
EL AP. AL ARCHIVO =       7
>@ <EOF>
>
>; LA SALIDA DE MPPG ES:
>
>PIP TI:=SALIDA.TMP
HOLA
A1A   A2A    A3A
HOLA TODOS
B1B   B2B    B3B
HOLA TODOS LO HUMANOS
C1C   C2C    C3C
ADIOS TODOS LOS HUMANOS QUE ME ESCUCHARON
>
```

3.MODO DE TRABAJO.

MPPG trabaja con un caracter de advertencia (tambien llamado de senal), el cual permite al preprocesador reconocer rapidamente si el texto es parte del lenguaje base o una palabra reservada de MPPG. Este caracter de advertencia (senal) permite que el tiempo de procesado sea menor ya que unicamente se tomaran en cuenta las palabras del texto fuente que empiecen con dicho caracter, mientras que las otras seran pasadas al archivo de salida tal cual.

4.CAMBIO DE CARACTER DE CONTROL.

En los ejemplos antes vistos, el caracter se senal era '%', pero podria cambiarse dependiendo del tipo de sistema y aun del lenguaje de programacion utilizado.

El cambio del caracter de senal se puede hacer ya sea mofificando el preprocesador utilizando un editor de textos o modificando los caracteres de advertencia al tiempo de preprocesado (dinamicamente).

MPPG pregunta al usuario por la terminal los caracteres que el desea utilizar, teniendo tambien sus propios caracteres previamente establecidos (%,~,[,]).

A los preprocesadores que utilizan un caracter de senal se les conoce como preprocesadores de modo advertencia. Existen preprocesadores llamados de modo libre, los cuales no necesitan ningun caracter de senal.

La desventaja del modo advertencia con respecto al modo libre es obviamente el uso de los caracteres de senal, los cuales pueden ser parte del conjunto de caracteres validos de algun lenguaje de programacion. Esta desventaja se ve reducida considerablemente con la opcion que se le da al usuario de poder escoger sus propios caracteres de senal.

Para los usuarios de un preprocesador o Macro-expansor es mas comodo utilizar un preprocesador en modo libre, ya que asi no necesita estar teniendo cuidado con los caracteres de advertencia. Esto que parece una desventaja a simple vista, al realizar el preprocesador, se convierte en una gran ventaja ya que cada identificador del programa fuente tendra que ser buscado en la tabla de nombres de Macros ya definidos. Todo esto desemboca en un mayor tiempo de procesado, ya que todos y cada uno de los identificadores y palabras reservadas del texto fuente tendran que ser verificadas en la tabla de nombres de Macros.

5.FORMAS DE MANEJO DEL TEXTO.

Existen dos formas de tratar el texto fuente, ya sea por caracteres o por atomos. El tratar el texto por caracteres consiste en ir leyendo del archivo fuente caracter por caracter y asi irlos examinando, de tal forma que dichos caracteres seran pasados tal cual al archivo de salida, a menos que el caracter leido sea un caracter especial (advertencia). En el momento de reconocer el caracter especial se van leyendo los caracteres subsecuentes para determinar la palabra reservada.

La forma de tratar el texto llamada por atomos consiste en leer el texto fuente caracter por caracter y construir elementos constituidos por caracteres validos ya sea para palabras reservadas o para identificadores y numeros. Estos atomos son construidos por un analizador lexico (scanner), de tal forma que el analisis del texto sea sobre dichos atomos.

MPPG trabaja en base a atomos ya que en el modo de caracter por caracter, el propio procesador tendria que ir examinando el texto fuente. Este examen del texto hecho por el procesador no permite modificaciones faciles, ya que si se desea cambiar la estructura de los identificadores (cambiar por un lenguaje base extrano) habria que modificar al propio procesador en varios de sus modulos. Pero usando el modo de atomos la modificacion consistiria unicamente en cambiar el analizador lexico.

Algo que hay que mencionar a favor del modo de caracter por caracter es la velocidad, este modo es mas rapido que el de atomos ya que no se pierde tiempo en la construccion de atomos.

6.MANEJO DE COMENTARIOS.

Una caracteristica importante y muy poderosa de algunos Macro-procesadores es el poder reconocer cuando el texto leido es parte de un comentario. Esto se realiza examinando los caracteres del texto fuente que son el inicio de los comentarios, por ejemplo, en los lenguajes Ensambladores el caracter de comentario comunmente es el caracter ';'. Ahora bien, es claro que no es necesario examinar el texto fuente que corresponde a comentarios, ayudando en el ahorro de tiempo de procesado, ya que cuando se a detectado que el texto es comentario, todo el texto fuente es pasado directamente al archivo de salida y este proceso se detiene hasta que se encuentre el caracter de fin de comentario. Sin embargo este proceso tiene el inconveniente de que cada lenguaje de programacion tiene sus propias reglas para los comentarios, entonces, la realizacion de Macro-procesadores se hace bastante compleja.

A modo de ejemplificar este problema, tomemos tres lenguajes de programacion de uso comun: FORTRAN, PASCAL, ENSAMBLADOR.

Las formas de definir comentarios en estos lenguajes difieren mucho ya que mientras en unos existen formatos fijos en otros existen campos especificos para los comentarios, mas aun, el inicio de comentarios no siempre esta determinado por un solo caracter, como es el caso de PASCAL.

Estos problemas redundan en una realizacion bastante complicada y de ninguna manera general, por lo tanto, el diseno de MPPG no tiene la capacidad de reconocer comentarios. El hecho de que MPPG no reconozca comentarios hace que los usuarios deban tener mucho cuidado de que en sus textos de comentarios no haya llamadas a funciones de MPPG ni Macro-llamadas.

7.DEFINICIONES DE MACROS.

MPPG trabaja en modo de advertencia y para definir un Macro se escribe:

```
%MACRO NOMBRE(arg1,arg2,...,argn)
{
cuerpo del macro
}
%ENDM
```

En donde se podran notar las palabras reservadas %MACRO y %ENDM, las cuales estan precedidas por el caracter de advertencia '%'.

Al definir un Macro (%MACRO), el nombre de dicho Macro(NOMBRE) es guardado en el directorio de nombres de Macros, el cual contendra todos los nombres de los Macros que hayan sido definidos.

A los identificadores arg1,arg2,...argn se les llama parametros formales y seran guardados en un area de almacenaje temporal (buffer) asociada al nombre del Macro.

El cuerpo del Macro sera almacenado en un archivo de Macros, donde permanecera, para poder ser accesado cuando se hasa un llamado a dicho Macro.

El cuerpo del Macro no sera almacenado tal cual fue puesto por el usuario, sino que, el preprocesador sustituira (en el cuerpo del Macro) los parametros formales por un caracter de advertencia (senal) y un numero. El numero dependera de la posicion del parametro formal dentro de la lista de parametros formales dados en la definicion.

MPPG utiliza en los parametros formales como ya se describio el caracter '^', este caracter de senal es distinto al usado para determinar las palabras reservadas o pseudo-instrucciones (%), ya que en caso contrario podria existir la posibilidad de confusion en el preprocesado. Este nuevo caracter de advertencia no debe ser del conjunto de caracteres validos para la aplicacion del Macro-procesador. Esta restriccion que de cierta manera es una desventaja, MPPG la resuelve permitiendo al usuario

definir sus propios caracteres de advertencia y de esta forma en el caso de que el usuario necesite cambiar los caracteres de advertencia lo pueda hacer modificando a MPPG o dinamicamente.

Para modificar dinamicamente los caracteres de advertencia, MPPG hace una pregunta por la terminal al usuario si es que desea efectuar algun cambio.

Dicho lo anterior el argi equivale a '~i' en el cuerpo del Macro en donde '~' es el caracter de advertencia e 'i' es la i-esima posicion dentro de la lista de argumentos formales (a '~i' le llamaremos argumento de advertencia numero i).

La sustitucion de los parametros formales por argumentos de advertencia permitira una mejor y mas rapida sustitucion. La justificacion es la siguiente:

a) El procesado se hace mas rapido (o sea la sustitucion de los parametros actuales por los argumentos de advertencia) debido a que el procedimiento que sustituye busca unicamente los caracteres de advertencia, y todo el texto es pasado directamente sin ser examinado. Si se piensa en la posibilidad de no usar caracteres de advertencia entonces se tendria que ir preguntando por cada identificador que fuera parte del cuerpo del Macro para ver si es o no parametro formal, lo cual evidentemente se resume en mas tiempo de procesado (este esquema es un caso particular del paso de parametros por palabra clave [DONOVAN-1]).

b) El uso de argumentos de advertencia es mejor en el sentido de que implica menor posibilidad de error para el usuario debido a que el cuerpo del Macro estando ya definido no contiene los identificadores tal cual fueron definidos (contiene los argumentos de advertencia), de esta forma el usuario puede utilizar como parametro actual un identificador que haya sido utilizado como parametro formal.

Podria pensarse el hecho de que el usuario de MPPG colocara el mismo los argumentos de advertencia dentro del cuerpo del Macro, sin embargo seria muy engorroso y sobre todo tenderia a aumentar la posibilidad de error.

Las Macro-definiciones son terminadas con la palabra %ENDM como se muestra en el ejemplo anteriormente descrito.

8.ANIDACION EN MACRO-DEFINICIONES.

Un punto importante de MPPG es el de admitir Macro-definiciones dentro de Macro-definiciones. Esto es, que en el cuerpo de una Macro-definicion pueda ir otra Macro-definicion, la cual seria 'propia' de la que la contiene, un ejemplo seria:

```
%MACRO M1(X)
------------
------------
        %MACRO M2(Y)
        ------------
        ------------
        %ENDM
------------
%ENDM
```

Notese la importancia ya que efectuando un llamado al Macro M1 (externo) se obtiene la definicion de M2 (interno). Esto es debido a que el cuerpo de una Macro-definicion es pasado tal cual al archivo de Macro-definiciones, es decir, no se le hace ningun tipo de analisis.

Cuando el cuerpo del Macro externo esta en el archivo de almacenamiento temporal mantiene la Macro-definicion que contiene dentro de su cuerpo y asi al efectuarse un llamado al Macro externo, se toma el cuerpo del Macro y se va expandiendo, pero si en el texto que se va expandiendo hay una Macro-definicion, entonces dicha Macro-definicion pasa a formar parte de los Macros definidos, es decir se le toma en cuenta para poder ser expandido como cualquier Macro-definicion.

Esta caracteristica de MPPG sera realizada en primera instancia en la secuencia de que por cada llamada al Macro externo se efectue una definicion del Macro interno, permitiendo mayor flexibilidad al usuario. Esto es debido a que el Macro externo podria contener varias Macros de tal forma que con expansion condicional se pudiera escoger entre dichos Macros cual seria el definido. Ademas de poder modificar cierto texto del macro interno, todo esto evidentemente en tiempo de preprocesado.

9.MACRO-LLAMADAS.

Para realizar una Macro-llamada (tambien conocida como Macro-expansion) es necesario llamar al Macro por su nombre utilizando el caracter de advertencia de la siguiente forma:

```
%NOMBRE(A,B,C...,Z)
```

En donde se puede ver el caracter de advertencia. Se usa el caracter de advertencia en la llamada para facilitar la expansion ya que es mas rapida la busqueda del identificador en la tabla donde estan almacenados los nombres de todos los Macros.

El identificador para el nombre, asi como los identificadores comunes, seran de la forma: el primer caracter sera alfabetico seguido de caracteres alfanumericos. La longitud de los identificadores esta determinada por la constante MAXLON (ver siguiente capitulo).

En el llamado al Macro se dan los parametros actuales (a,b,c,...,z) y la relacion de los parametros actuales con los parametros formales es posicional, es decir, el primer parametro actual corresponde al primer parametro formal; el segundo parametro actual corresponde al segundo parametro formal y asi sucesivamente.

Si los parametros actuales son menor en numero que los parametros formales, entonces los parametros formales restantes tendran asignado el valos nulo. Tambien es valido que el usuario pase como parametro actual la cadena nula. Hay que hacer notar que no importa si en una Macro llamada hay mas parametros actuales que parametros formales ya que MFPG ignora a los parametros actuales sobrantes.

Si alguno de los parametros actuales consiste de un conjunto de de caracteres conteniendo cualquier tipo de separador o caracteres especiales, entonces debera ser colocado dentro de parentesis cuadrados.

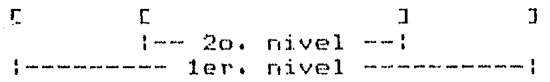
Por ejemplo, si el parametro actual es: I:=I+1 entonces un llamado podria ser:

%NOMBRE(A,B,[I:=I+1],D)

Igualmente es posible que un parametro actual contenga entre sus caracteres el parentesis cuadrado. Por ejemplo, supongase el siguiente Macro-llamado:

%NOMBRE(A,[IF A[I]<0 THEN],C,D)

Esto es permitido por MPPG ya que toma los parentesis cuadrados por niveles, de tal forma que en la Macro-expansion no toma en cuenta los parentesis cuadrados del primer nivel, como se ilustra en la siguiente figura:



Para darle a MPPG mas poder y evidentemente mas aplicacion, se realizo el paso de parametros de tal manera que pudiera pasarse como parametro actual una Macro-llamada.

A continuacion se da un ejemplo concreto:

```
>
>
>; EJEMPLO DE MACRO EXPANSION
>
>
>PIP TI:=MONITOR.MAC
;       RUTINA MONITOR
;       MUESTRA LOS REGISTROS
;
;       DIRECTIVAS DE ENTRADA/SALIDA
        .MCALL QIOW$, DIR$
;       AREA DE DATOS
REG0:   .WORD
REG1:   .WORD
REG2:   .WORD
BUFSAL: .BLKW  6
ESC:    QIOW$   IO.WVB,5,1,,,,<BUFSAL,6,40>
        %MACRO SACANUM(X)
        MOV  X,R1
        MOV  #1,R2
        MOV  #BUFSAL,R0
        CALL $CBOMG
        DIR$ #ESC
        %ENDM
MONITOR:JSR  PC,$SAVAL
        MOV  R0,REG0
        MOV  R1,REG1
        MOV  R2,REG2
        %SACANUM(REG0)
        %SACANUM(REG1)
        %SACANUM(REG2)
        %SACANUM(R3)
        %SACANUM(R4)
        %SACANUM(R5)
        RTS  PC
        .END MONITOR
>
>; SE LLAMA AL MACRO PROCESADOR
>
>@A
>PIP *.TMP;*/DE
>RUN MPPG
CAMBIO DE CARACTER DE ADVERTENCIA ?
N
USARA SU PROPIA BIBLIOTECA?
N
DEME EL NOMBRE DEL ARCHIVO FUENTE
MONITOR.MAC
EL DIRECTORIO DE MACROS ES:
EL MACRO SE LLAMA SACANUM          Y SU AP. AL ARCHIVO ES:        1
 TABLA DE SIMBOLOS DEF.
EL AP. AL ARCHIVO =        7
>@ <EOF>
>
```

```
>; LA SALIDA DE MPPG ES:
>
>
>PIP TI:=SALIDA.TMP
;       RUTINA MONITOR
;       MUESTRA LOS REGISTROS
;
;       DIRECTIVAS DE ENTRADA/SALIDA
        .MCALL QIOW$, DIR$
;       AREA DE DATOS
REG0:   .WORD
REG1:   .WORD
REG2:   .WORD
BUFSAL: .BLKW  6
ESC:    QIOW$   IO.WVB,5,1,,,,<BUFSAL,6,40>
MONITOR:JSR  PC,$SAVAL
        MOV  R0,REG0
        MOV  R1,REG1
        MOV  R2,REG2
        MOV  REG0,R1
        MOV  #1,R2
        MOV  #BUFSAL,R0
        CALL $CBOMG
        DIR$ #ESC
        MOV  REG1,R1
        MOV  #1,R2
        MOV  #BUFSAL,R0
        CALL $CBOMG
        DIR$ #ESC
        MOV  REG2,R1
        MOV  #1,R2
        MOV  #BUFSAL,R0
        CALL $CBOMG
        DIR$ #ESC
        MOV  R3,R1
        MOV  #1,R2
        MOV  #BUFSAL,R0
        CALL $CBOMG
        DIR$ #ESC
        MOV  R4,R1
        MOV  #1,R2
        MOV  #BUFSAL,R0
        CALL $CBOMG
        DIR$ #ESC
        MOV  R5,R1
        MOV  #1,R2
        MOV  #BUFSAL,R0
        CALL $CBOMG
        DIR$ #ESC
        RTS  PC
        .END MONITOR
>
>
>
>
>
```

10.ANIDACION EN MACRO-LLAMADAS

MPPG permite anidacion en las llamadas a Macros, es decir, que dentro del cuerpo de un Macro se llame a otro Macro. Por ejemplo:

```
%MACRO M1(X)
IF X > 0 THEN X:=X-10
%ENDM
%MACRO M2(Y)
A=SEN(Y)
%M1(A)
%ENDM
BEGIN
%M2(30)
END.
```

Con lo cual, despues de la expansion quedaria:

```
BEGIN
A=SEN(30)
IF A > 0 THEN A:=A-10
END.
```

La anidacion es permitida por MPPG ya que despues de haberse reemplazado el parametro actual por el parametro formal, se examina la cadena resultante y si se encuentra el caracter de advertencia entonces se procede a examinar para saber si se trata de una Macro-definicion o de una Macro-llamada. Si se trata de una Macro-llamada se procede a la Macro-expansion del nuevo Macro-llamado. Y una vez terminado el Macro-llamado, prosigue con la Macro-expansion que habia sido detenida.

11.RECURSION.

MPPG admite recursion, es decir, permite que un Macro se llame a si mismo. Esto es permitido gracias a la posibilidad de manipular registros de medio ambiente, los cuales son construidos por MPPG y seran descritos en el siguiente capitulo. Sin embargo es necesario contar con expansion condicional, de tal manera que la recursion no sea un proceso infinito (en secciones posteriores se explicaran las pseudo-instrucciones de expansion condicional).

12. PROPOSICIONES DE ASIGNACION DIRECTA.

MPPG admite proposiciones de asignacion directa, mas especificamente, una proposicion de asignacion directa permite igualar un simbolo a un valor especifico.

Cuando una proposicion de asignacion directa es utilizada y el simbolo involucrado aparece por primera vez, el simbolo es colocado dentro de una tabla de simbolos definidos por el usuario.

Un simbolo definido por el usuario puede ser redefinido por una subsecuente proposicion de asignacion directa, por asignarle un nuevo valor al simbolo previamente definido.

El formato general de una proposicion de asignacion directa es:

```
%SIMBOLO = EXPRESION
```

En donde simbolo es una cadena de caracteres alfanumericos y el primer caracter debe ser alfabetico, ademas el simbolo debe ser precedido por el caracter de advertencia.

Las expresiones son combinaciones de terminos unidos por operadores binarios, las cuales son reducidas a un solo valor, es decir, son evaluadas. Los terminos deben ser de tipo entero.

Un termino es un componente de una expresion y puede ser un numero entero, un simbolo que fue definido por el usuario o bien un simbolo no definido. Un simbolo es definido si dicho simbolo esta en la tabla de simbolos definidos por el usuario.

Si el termino es un simbolo definido por el usuario entonces se toma su valor correspondiente de la tabla de simbolos definidos por el usuario. Y si el termino es es un simbolo que no haya sido definido (utilizado) previamente, toma el valor de cero.

Ejemplos de terminos:

```
3

%TABLA
```

Las expresiones son evaluadas de izquierda a derecha sin reglas de precedencia sobre los operadores, excepto que los operadores unarios toman precedencia sobre operadores binarios.

Un termino precedido por un operador unario es considerado a contener ese operador.

Un termino o expresion perdida sera interpretada como un cero. Un error en los operadores o en los terminos, terminara el analisis de la expresion. Por ejemplo la expresion:

```
10 + 5          20
```

Es evaluada como (10 + 5) ya que el primer caracter distinto de blanco, que esta despues del numero 5, no es un operador legal (+,-,*,/).

Los espacios dentro de las expresiones son ignorados.

Ejemplos de proposiciones validas:

```
%CONTADOR = 3

%INDICE = %BASE * 3 + 5
```

13. EXPANSION CONDICIONAL.

MPPG admite tambien directivas o pseudo instrucciones de expansion condicional, que permiten al usuario incluir o excluir bloques de texto fuente durante el preprocesado. Con esto se obtiene una de las herramientas mas poderosas de MPPG.

La inclusion o exclusion del texto esta basada en la evaluacion y examen de las condiciones de estado, dentro del cuerpo del programa fuente. Esta capacidad permite obtener variaciones de un programa a ser generadas del mismo texto fuente.

El poder obtener variaciones de programas en base al mismo texto fuente es muy usado en la construccion de Sistemas Operativos. De esta forma un conjunto de bloques (los manejadores de dispositivos perifericos por ejemplo) pueden ser incluidos si es que la configuracion fisica del centro de computo lo necesita. Mas especificamente, supongase que se tiene un Sistema Operativo que tiene todos los manejadores de dispositivos posibles en forma de modulos.

Los Sistemas Operativos deben tener todos los manejadores para todos los posibles perifericos disponibles ya que el Sistema Opertivo debe ser lo mas general posible. Ahora bien, ya en un centro de computo especifico supongase que no se tiene impresora, entonces es claro que el manejador para la impresora no es necesario que este presente en el sistema ya que ademas de ocupar espacio fisico, es ilogico tenerlo porque nunca se utilizaria.

De igual manera existen otros modulos que deberan estar siempre en el texto fuente pero que dependiendo de la configuracion del sistema se escogera que modulos se usaran, por ejemplo, si es un Sistema Operativo mapeado o no mapeado a memoria; si se cuenta con memoria central suficiente o no; si se desean listados de los mapas generados al construirse el sistema o no se desean, y asi sucesivamente existen muchos elementos de decision los cuales tienen varios niveles de importancia pero que son necesarios para un mejor aprovechamiento de los recursos fisicos.

El formato de las directivas de expansion condicional es:

```
      %IFXX ARGUMENTO

       CUERPO

      %ENDC
```

DONDE:

XX Son dos caracteres que representan la condicion que debe ser satisfecha para que el bloque de texto fuente sea incluido.

Las condiciones admisibles estan definidas en la tabla 1.

ARGUMENTO Representa un argumento simbolico o expresion que sera examinada dependiendo de la condicional que fue especificada por XX.

CUERPO Representa el texto fuente que sera incluido o excluido dependiendo de si la condicion fue satisfecha o no.

%ENDC Determina el final del bloque de texto afectado

TABLA 1. Tabla de directivas de expansion condicional legales.

CONDICION	EXPANDE EL BLOQUE SI
EQ	La expresion es igual a cero.
NE	La expresion no es igual a cero.
GT	La expresion es mayor que cero.
LT	La expresion es menor que cero.
GE	La expresion es mayor o igual a cero.
LE	La expresion es menor o igual a cero.

Ejemplo:

```
            %IF     %CONT + 1
            WRITE(5,100)
      100   FORMAT(1X,'HOLA TODOS',//)
            %ENDC
```

Esta directiva de expansion condicional hara que el cuerpo sea parte del texto fuente si la expresion (%CONT + 1) es distinta de cero. Esto evidentemente despues de efectuar la evaluacion de la expresion.

Como se podra observar, con las seis directivas de expansion condicional se pueden obtener todas las posibles equivalencias, por ejemplo, supongase que un bloque de texto se desea que forme parte del texto fuente si el valor de un simbolo %TABLA es mayor que tres, es decir:

```
%TABLA > 3
```

Ahora bien, esto se puede escribir de la siguiente forma:

```
%TABLA - 3 > 0
```

lo cual nos lleva a:

```
%IFGT   %TABLA - 3

 CUERPO

%ENDC
```

14. COMBINACIONES DE EXPANSION CONDICIONAL, MACRO LLAMADAS Y MACRO DEFINICIONES.

Las directivas de expansion condicional pueden ser anidadas, por ejemplo, el siguiente bloque sera incluido como texto fuente si %FLAG1 y %FLAG2 son mayores que cero.

```
%IFGT   %FLAG1

%IFGT   %FLAG2

<CUERPO>

%ENDC

%ENDC
```

Notese ademas que dentro de algun bloque contenido en una directiva de expansion condicional puede ir una Macro definicion o una Macro llamada.

Ejemplo de una Macro definicion dentro del cuerpo de una directiva de expansion condicional:

```
%IFNE   %SIMB1

--------------

--------------

--------------

        %MACRO  M1

        ----------

        ----------

        %ENDM

--------------

%ENDC
```

Ejemplo de una Macro llamada dentro del cuerpo de una directiva de expansion condicional:

```
PROG1
-----------------
-----------------
%MACRO  M2
---------------------
---------------------
%ENDM
---------------
---------------
---------------
%IFLE   %SIMBOL2
---------------
---------------
%M2
---------------
%ENDC
---------------
END.
```

Mas aun, no hay ningun problema para utilizar directivas de expansion condicional dentro de una definicion macro, por ejemplo:

```
%MACRO  M1
-----------------
-----------------
-----------------
%IFGT   %NOM + 3
--------
--------
%ENDC
-----------------
-----------------
%ENDM
```

Las directivas de expansion condicional permiten al usuario utilizar macros recursivos de tal manera que el proceso no sea infinito.

15. ENTRADA / SALIDA CONTROLADA.

MPPG admite entrada / salida controlada, es decir, permite al usuario dar por la terminal un caracter o cadena de caracteres de tal manera que sea procesado como si fuera parte del texto fuente. Esta posibilidad de dar por la terminal una cadena, permite al usuario que en determinado momento (dinamicamente) pueda dar como texto de entrada una llamada a algun Macro, de tal forma que se puedan incluir algunas partes de texto que contengan Macro definiciones, (para mayor referencia ver: definiciones de macros dentro de macro definiciones), recordando que cuando existe anidacion en las Macro definiciones, el Macro que es interno no es definido sino hasta que el Macro externo es llamado.

La caracteristica de MPPG de admitir entrada / salida controlada va mas adelante ya que como cadena de entrada puede ir el cuerpo de un Macro. Esta posibilidad permite al usuario cambiar el texto de algun Macro dinamicamente o dar como texto de entrada el cuerpo de un Macro.

En algunos Macro procesadores como es el caso de TRAC, este proceso es interactivo, sin embargo podemos pensar en un proceso que sea interactivo pero que no sea totalmente conversacional, ya que esto tiende a confundir al usuario y a hacer mas lento el proceso entre otras cosas.

El tipo de MPPG es interactivo pero no totalmente conversacional y la argumentacion profunda es la siguiente:

a) Si el proceso es totalmente conversacional y el usuario da varias Macro definiciones o Macro llamadas, en distintos momentos del procesado, es evidente que puede haber confusion con las Macro definiciones dadas por la terminal. Esta posibilidad de error puede costarle al usuario el tener que repetir todo el proceso desde un principio, y si pensamos que el proceso consta del uso de bibliotecas de Macros, cambio de caracteres de senal, etc.; entonces es un costo bastante alto para el usuario.

b) Es evidente que el usuario, por muy rapido que sea al teclear nunca va a poder competir con la velocidad de la computadora, y si debe estarle dando texto desde la terminal en varias ocasiones, resulta que todo esto redunda en que el tiempo de procesado se vera enmarcado por la velocidad del usuario y por la cantidad de texto que se desee dar como entrada.

c) La entrada controlada es en estos momentos poco utilizada, esta afirmacion esta basada en la experiencia obtenida ya que existen pocos sistemas que lo permiten y poca gente en el desarrollo de este tipo de sistemas.

```
>
>; EJEMPLO DE ENTRADA/SALIDA CONTROLADA.
>
>
>PIP TI:=EJEMES.TXT
%ETI=10+02
C MAIN
%DEFTTY
C LINEA1
%M1(X)
C LINEA2
%EXPTTY
C LINEA3
>
>; SE LLAMA A MPPG.
>
>@A
>PIP *.TMP;*/DE
>RUN MPPG
CAMBIO DE CARACTER DE ADVERTENCIA ?
N
USARA SU PROPIA BIBLIOTECA?
N
DEME EL NOMBRE DEL ARCHIVO FUENTE
EJEMES.TXT
M1(A)
XX A XX
XX A XX
%ENDM

M1(Y)

 EL DIRECTORIO DE MACROS ES:
EL MACRO SE LLAMA M1                Y SU AP. AL ARCHIVO ES:        1
 TABLA DE SIMBOLOS DEF.
 EL SIMBOLO ES:ETI                Y SU VALOR ES:       12
EL AP. AL ARCHIVO =        4
>@ <EOF>
>
>; LA SALIDA DE MPPG ES:
>
>
>PIP TI:=SALIDA.TMP
  C MAIN
C LINEA1
XX X XX
XX X XX
C LINEA2
XX Y XX
XX Y XX
C LINEA3
>
>
>
```

16. FUNCIONES PREDEFINIDAS.

MPPG tiene funciones implicitas, es decir, funciones que son parte de MPPG. Esta caracteristica permite mayor velocidad en el procesado del texto ya que MPPG manipula el texto directamente. Ejemplos de las funciones predefinidas son las directivas de expansion condicional, las cuales fueron ya descritas en la seccion 13.

En esta seccion se describiran las funciones predefinidas PUSH y POP.

Debido a la amplia aplicacion de MPPG se penso en darle al usuario la posibilidad de manejar una estructura de datos en tiempo de procesado. Y pensando en la estructura mas acorde a las aplicaciones de MPPG se construyo una pila.

Una pila es una lista lineal que se caracteriza porque el primer elemento en entrar a la lista es el ultimo en salir.

Graficamente la pila se veria de la siguiente forma:

Como se podra observar las inserciones y las supresiones se hacen por el tope. En el siguiente capitulo (REALIZACION) se describira completamente la estructura ya que en esta seccion se vera la forma de utilizar las funciones unicamente.

La pila sera manejada por el usuario haciendo llamadas a las funciones predefinidas que son:

PUSH Mete el valor del primer simbolo que haya sido definido por el usuario a la pila y actualiza el apuntador a dicha pila.

POP Actualiza el apuntador y saca el valor que esta en el tope de la pila para depositarlo como el valor del segundo simbolo que fue definido por el usuario.

El uso de esta estructura permitira al usuario la anidacion de proposiciones o unidades sintacticas definidas por el mismo. Hay que hacer notar que el manejo de la pila sera hecho por el usuario en tiempo de procesado.

A continuacion se muestra un ejemplo del uso de la pila, en este ejemplo se realiza la proposicion WHILE en base del uso de las funciones predefinidas. Hay que recordar que en FORTRAN no se tiene dicha proposicion, ademas de que se permite la anidacion de la proposicion.

CAPITULO IV REALIZACION.

1. INTRODUCCION.

El departamento de Matematicas de la Facultad de Ciencias de la U.N.A.M. cuenta actualmente (1982) con el siguiente equipo de computo en sus laboratorios de computacion:

a) 3 terminales conectadas a la computadora B-6700.

b) 1 minicomputadora PDP-11/34.

c) 1 minicomputadora PDP-11/03.

d) 1 microcomputadora motorola-6800.

e) 1 microcomputadora INTEL 8080.

La disponibilidad del equipo puede ser sopesada con la siguiente informacion:

a) Existen tres terminales conectadas a la B-6700 y la mayoria de los profesores de tiempo completo trabajan solamente en dicha computadora, ademas de que no todos los estudiantes tienen acceso a dicho equipo.

b) Las microcomputadoras y la minicomputadora PDP-11/03 tienen sistemas para un usuario unicamente, ademas de contar con FORTRAN como lenguaje de alto nivel.

c) La PDP-11/34 es la unica de todas las computadoras (excluyendo obviamente a la B-6700) que tiene sistema operativo multi-usuario, ademas de un equipo periferico que da facilidad de uso en cualquier momento. La PDP-11/34 cuenta con dos unidades de disco, seis terminales, lectora de tarjetas y una impresora.

Esta informacion dada a grandes rasgos, muestra de una manera superficial el porque se escogio a la minicomputadora PDP-11/34 para hacer el presente trabajo. Podria darse una argumentacion profunda y bien fundamentada sobre esta decision, pero considero que se sale de los objetivos del presente escrito.

La minicomputadora PDP-11/34 tiene los siguientes compiladores:

a) FORTRAN

b) PASCAL

c) 'C'

Se opto por el lenguaje PASCAL ya que debido a las caracteristicas de MPPG es necesario el manejo de texto de una manera fluida, es decir, con variables acordes al tipo de informacion que se va a manejar. Esta informacion es esencialmente caracteres.

FORTRAN es un lenguaje que no fue construido para el manejo de texto, por lo tanto, su uso para la realizacion de MPPG tenderia a ser deficil y obscura. Mas aun, recordando que MPPG admite recursion, es evidente que la mejor manera de manejar problemas recursivos es con un lenguaje recursivo.

Dadas las anteriores argumentaciones, podria pensarse en el compilador 'C', ya que este lenguaje cumple con los requisitos anteriormente mencionados. Sin embargo, si pensamos que 'C' es un lenguaje relativamente nuevo, en donde relativamente nuevo quiere decir que pocos centros de computo lo tienen, esto nos lleva a que el uso de este tipo de software se restrinja. Esta restriccion va en contra de los objetivos basicos de MPPG ya que lo que se deseaba era un software de una aplicacion muy amplia, como se mostro en el primer capitulo.

2. DESCRIPCION DEL PROGRAMA.

Teniendo determinado el equipo y el lenguaje a usar en la realizacion de MPPG, debemos pasar a tomar en cuenta algunas consideraciones importantes ya que de estas dependera directamente la estructura de MPPG.

a) Debido a que MPPG trabaja en modo de advertencia hay que pensar en un conjunto de caracteres de senal que no sea muy utilizado.
Estos caracteres en principio seran '%' y '~'.
Recordando que debera ser posible el cambio de dichos caracteres.

b) Se debera construir un Analizador Lexico de tal manera que con cada llamada a dicho modulo nos de un atomo.
Este Analizador lexico evidentemente sera la parte central de MPPG ya que todos los demas modulos que necesiten obtener algun simbolo del texto de entrada deberan hacerlo por medio de este Analizador Lexico.
Esto tiene la gran ventaja de que los identificadores estan determinados por el Analizador, y si en algun momento se desea que los identificadores sean de otro tipo, sera posible con modificar al Analizador unicamente.

c) Una de las caracteristicas importantes de MPPG es la anidacion en las definiciones. Este problema sera resuelto con un contador de niveles, es decir, como el texto dentro del Macro debera ser almacenado tal cual, entonces todo lo que hay que hacer es ir contando el numero de apariciones de la palabra %MACRO y verificar que es el mismo numero de %ENDM.

d) La caracteristica mas importante es obviamente la recursion , ademas de ser la mas compleja de realizar.
La recursion trae consigo mas consideraciones a tomar, como son: expansion condicional, uso de metavariables, expresiones con metavariables, actualizacion de metavariables, etc., etc.
El uso de metavariables es esencial para la expansion condicional ya que se efectuan evaluaciones de tal manera que la condicion especificada sea verdadera o

falsa. A las metavariables se les puede asignar un valor constante o una expresion que contenga metavariables y constantes.
Debido al uso de metavariables, es necesario una estructura de datos que permita almacenar los identificadores asi como su valor. Mas aun, es necesario un evaluador de expresiones, o sea una funcion que pueda accesar y modificar metavariables.
La expansion condicional es la parte medular de la recursion ya que esta sera la que indique una llamada a si mismo o un fin de macro expansion. Este tipo de caracteristicas de MFPG debera ser tal que permita la mayor flexibilidad con el menor numero de funciones. como se menciono en capitulos anteriores, es suficiente con seis (NE, EQ, LE, LT, GE, GT).

El formato de la expansion condicional sera:

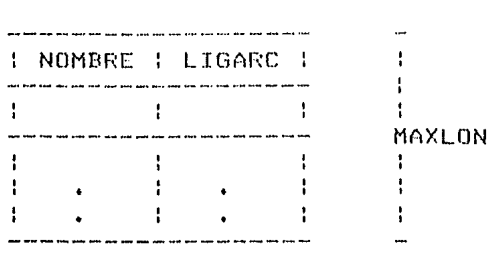
```
%IF CONDICION   EXPRESION
TEXTO
%ENDC
```

De tal manera que al reconocerse la expansion condicional, se evalue la expresion y si la condicion es verdadera, se procede a tomar en cuenta el texto fuente. Y si la condicion es falsa, entonces el texto no es tomado en cuenta para nada.

e) Como ultima consideracion importante hay que recordar que MFPG no debe depender del numero de Macros definidos, ni de la longitud de estos, por lo tanto no es conveniente que el area de almacenamiento temporal este dentro del programa (arreglos), ya que se correria el riesgo de que dicha area se llene produciendo un error fatal y dificil de resolver en MFPG.
La mejor forma de resolver este problema es la creacion de un archivo auxiliar en algun dispositivo de memoria secundaria como disco, cinta magnetica, etc.

Tomando en cuenta todas las anteriores consideraciones, se disenaron las siguientes estructuras de datos:

1.Una tabla llamada DIRMAC que contenga los nombres de los Macros que vayan siendo definidos por el usuario. Esta tabla tiene la siguiente forma:



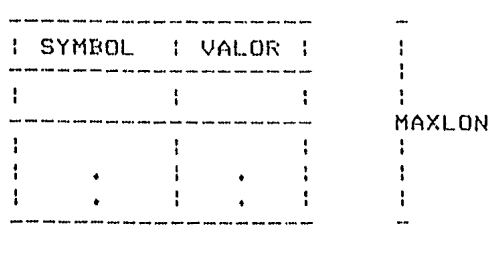
Es decir, es un arreglo de registros cuya dimension es MAXLON. Los registros tienen los siguientes campos:

a) El nombre del Macro (NOMBRE).

b) El apuntador al archivo auxiliar que contiene el cuerpo del Macro asociado a dicho nombre (LIGARC).

2.Una tabla llamada TABSYM que contenga los nombres de las variables que el usuario defina (metavariables).

Esta tabla tiene la siguiente forma:

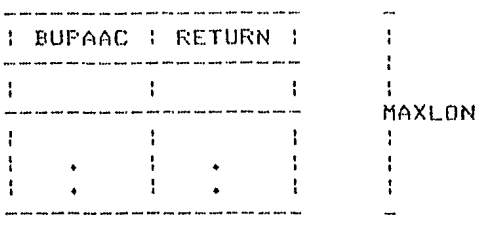


Es decir, un arreglo de registros cuya dimension es MAXLON . Los registros tienen los siguientes campos:

a) El nombre del simbolo (SYMBOL).

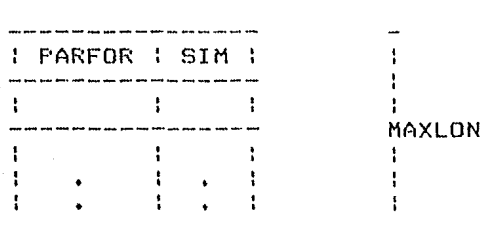
b) El valor asociado a dicho simbolo (VALOR).

3. Una tabla llamada REGMA que contendra la informacion necesaria para la anidacion. Esta tabla tiene la siguiente forma:



Es decir, un arreglo de registros cuya dimension es MAXLON. Los registros tienen los siguientes campos:

a) Un area de almacenamiento temporal donde seran guardados los parametros actuales (BUFAAC). Esta area es a su vez un arreglo de registros de la siguiente forma:



Y consta de los siguientes campos:
i) Area donde es almacenado el parametro actual(PARFOR).

ii) Area donde se almacena el tipo de simbolo que es, usando los mismos identificadores que en el modulo SCANNER(analizador lexico).

Todo este campo (BUFAAC) sera el que almacene a todos los parametros actuales que sustituiran a los parametros formales cuando se este expandiendo un Macro.

b) Un apuntador (RETURN) al archivo auxiliar que determinara el lugar al cual hay que retornar para seguir examinando el texto. Recordando que el examen del texto es detenido al encontrar una Macro llamada, y una vez satisfecha esta llamada hay que continuar con la expansion que se estaba realizando. Todo este trabajo es sobre el archivo auxiliar.

4.Otra tabla importante es PARFO, que almacenara los parametros formales mientras se efectua la busqueda de estos en todo el cuerpo del Macro para efectuar la sustitucion de los parametros formales por los parametros de advertencia (senal). PARFO es una tabla de longitud MAXLON y la referencia a alguno de los parametros se hace por medio del indice.

El hacer uso de parametros de senal permite que el usuario pueda utilizar como parametro actual al mismo identificador que uso como parametro formal. Ademas de esta ventaja para el usuario, se podra observar que en la realizacion del programa las cosas se facilitan ya que las comparaciones de los identificadores para encontrar a los parametros formales se hacen al principio (en la definicion) y no al momento de expander el Macro, lo cual nos complicaria mas la estructura del modulo EXPMAC.

5.Para terminar esta descripcion se daran los nombres de las principales variables con un breve comentario.

STACK - Es la pila que utilizara el usuario con las funciones PUSH y POP. Su longitud es MAXLON.

SP - Apuntador a la pila del usuario.

APRGMA- Apuntador al reg. de medio ambiente.

APDIR - Apuntador al directorio de Macros.

APTASY- Apuntador a la tabla de simbolos.

APTAFO- Apuntador a la tabla de parametros formales.

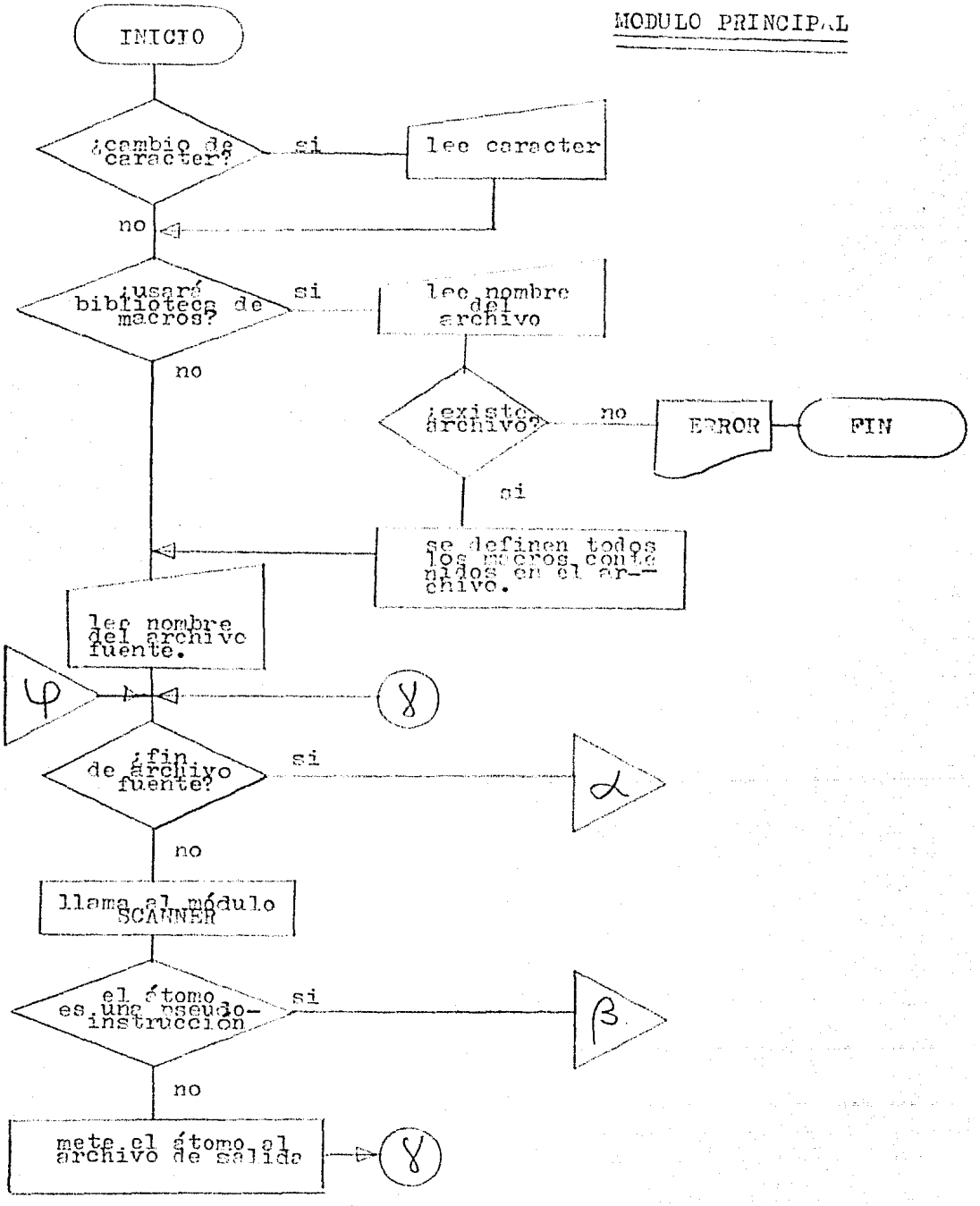
APFILE- Apuntador al archivo auxiliar para saber a partir de donde empieza el cuerpo del Macro. Dicho valor sera colocado en el campo LIGARC de la estructura DIRMAC

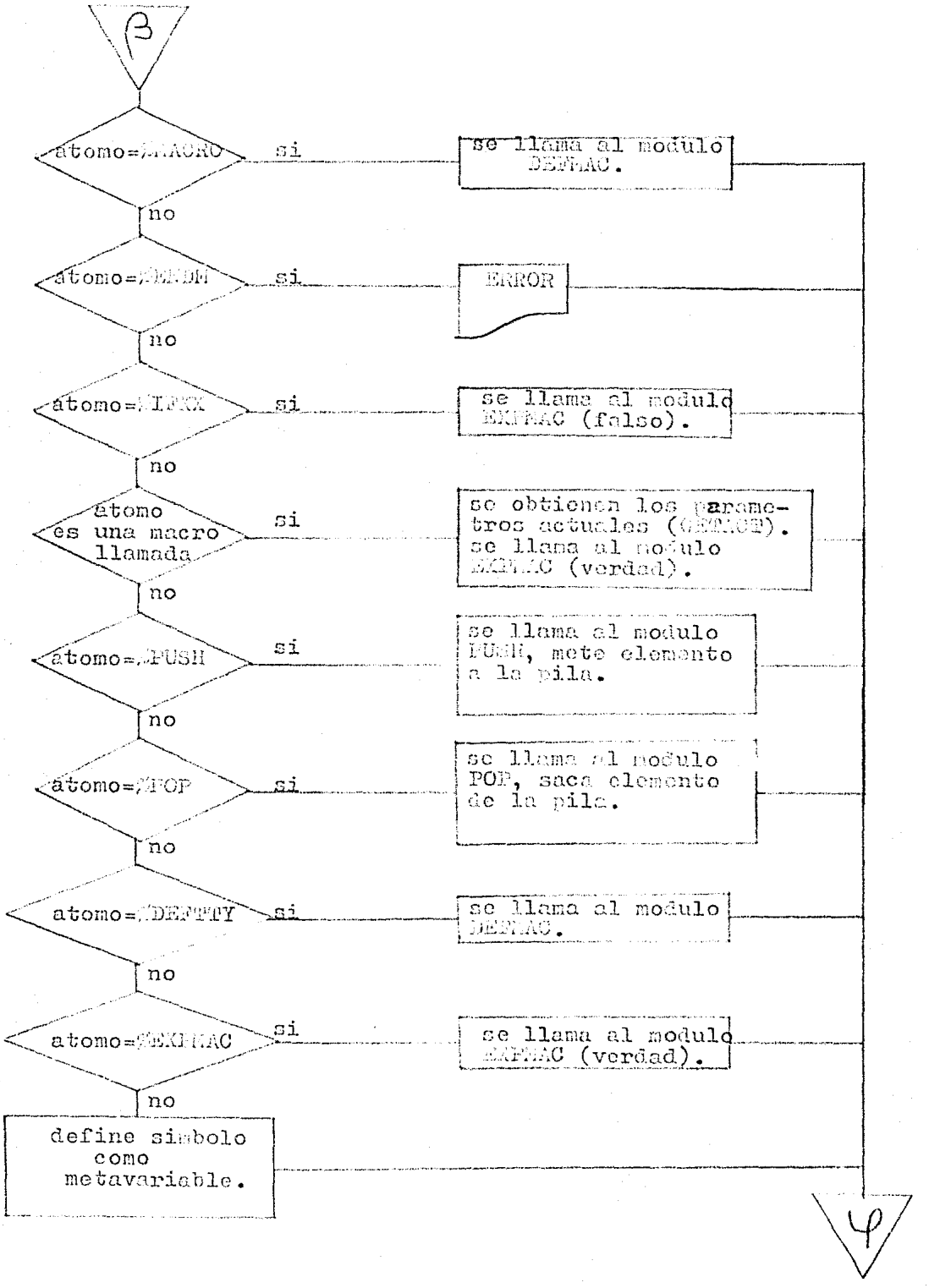
RESUL - Variable global en donde quedara almacenado el valor resultante al realizar alguna evaluacion por el modulo EVALUA.

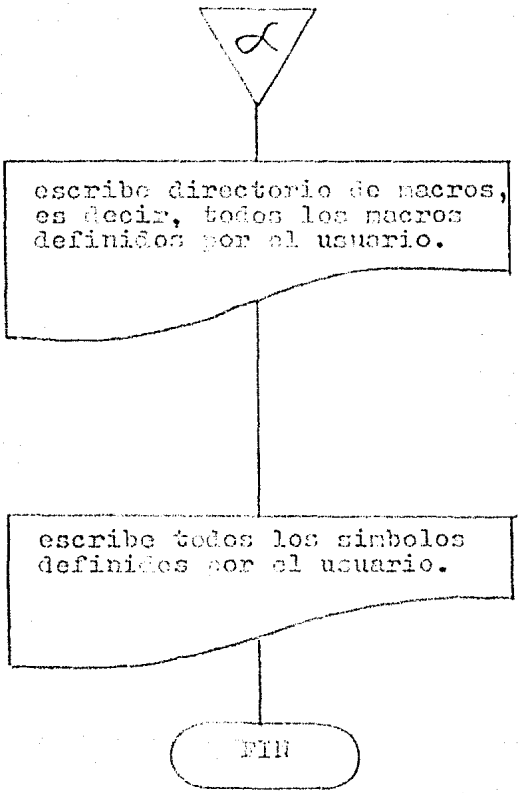
SPACES- Numero de espacios que anteceden al atomo.

APBUAC- Apuntador al area de almacenamiento de los parametros actuales.

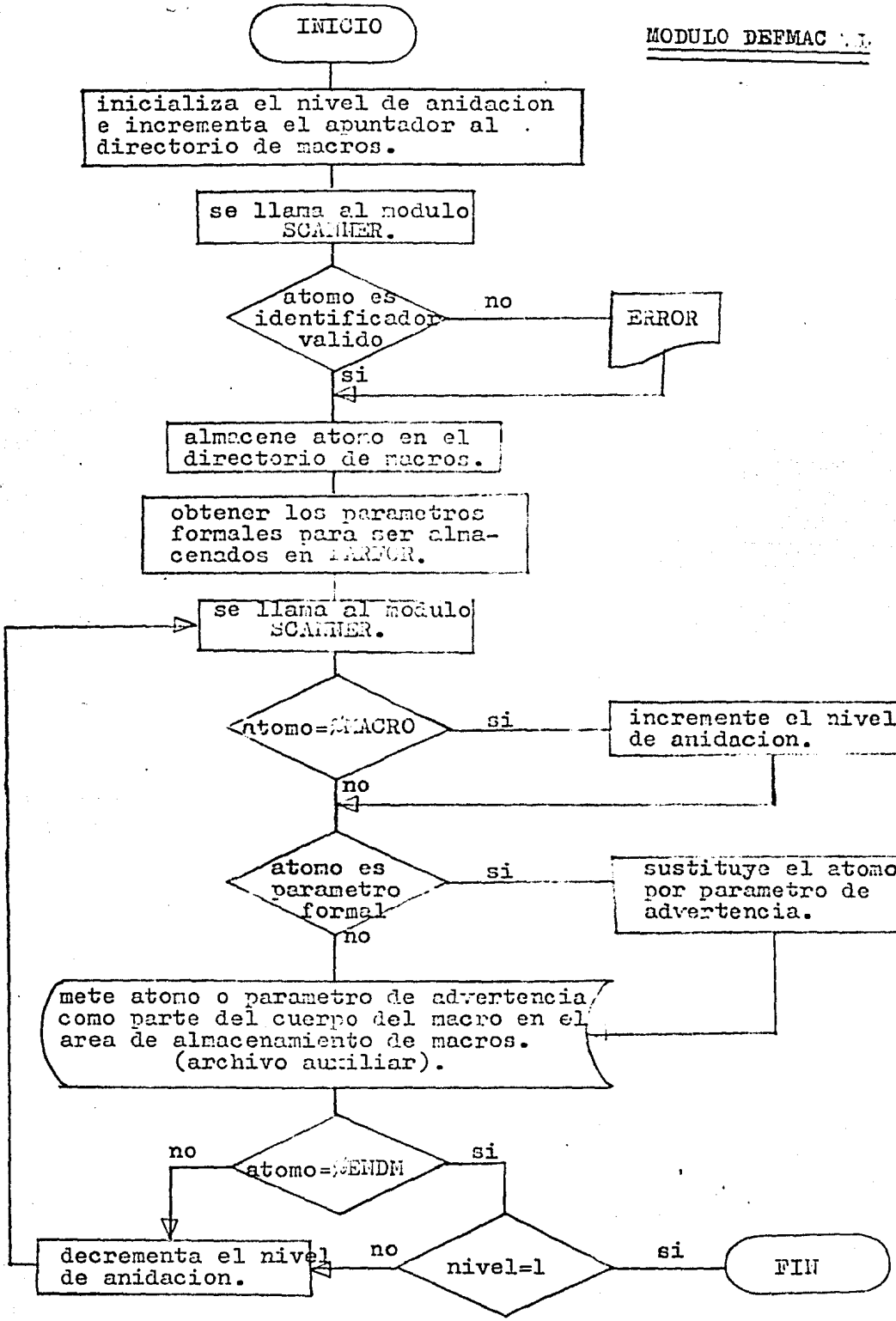
MODULO PRINCIPAL



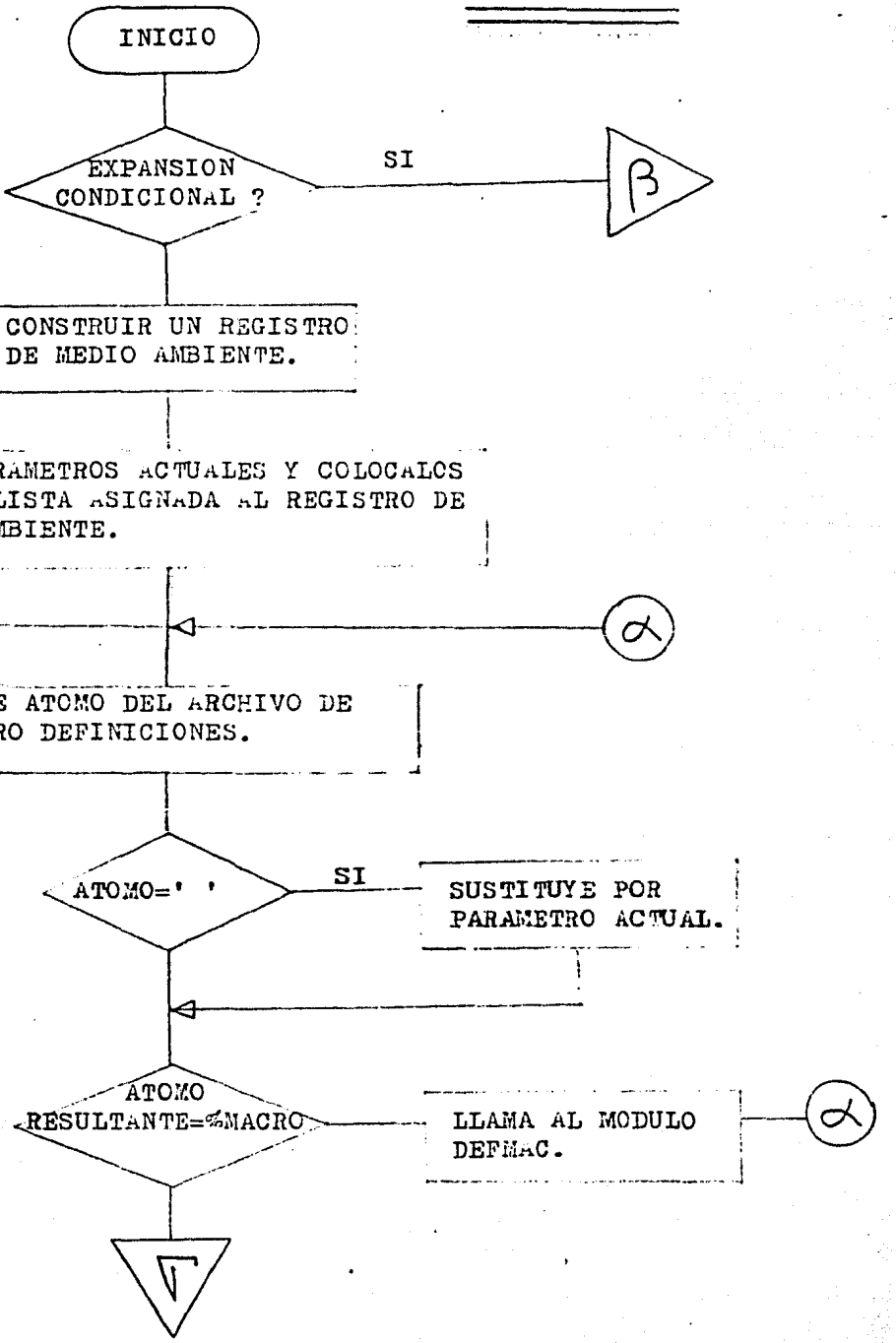

β
atomo=/MACRO
si
se llama al modulo DEFMAC.
no
atomo=/ENDM
si
ERROR
no
atomo=/IFCX
si
se llama al modulo EXPMAC (falso).
no
atomo es una macro llamada
si
se obtienen los parametros actuales (/PARAM).
se llama al modulo EXPMAC (verdad).
no
atomo=/PUSH
si
se llama al modulo PUSH, mete elemento a la pila.
no
atomo=/POP
si
se llama al modulo POP, saca elemento de la pila.
no
atomo=/DEFTTY
si
se llama al modulo DEFMAC.
no
atomo=/EXPMAC
si
se llama al modulo EXPMAC (verdad).
no
define simbolo como metavariable.
γ

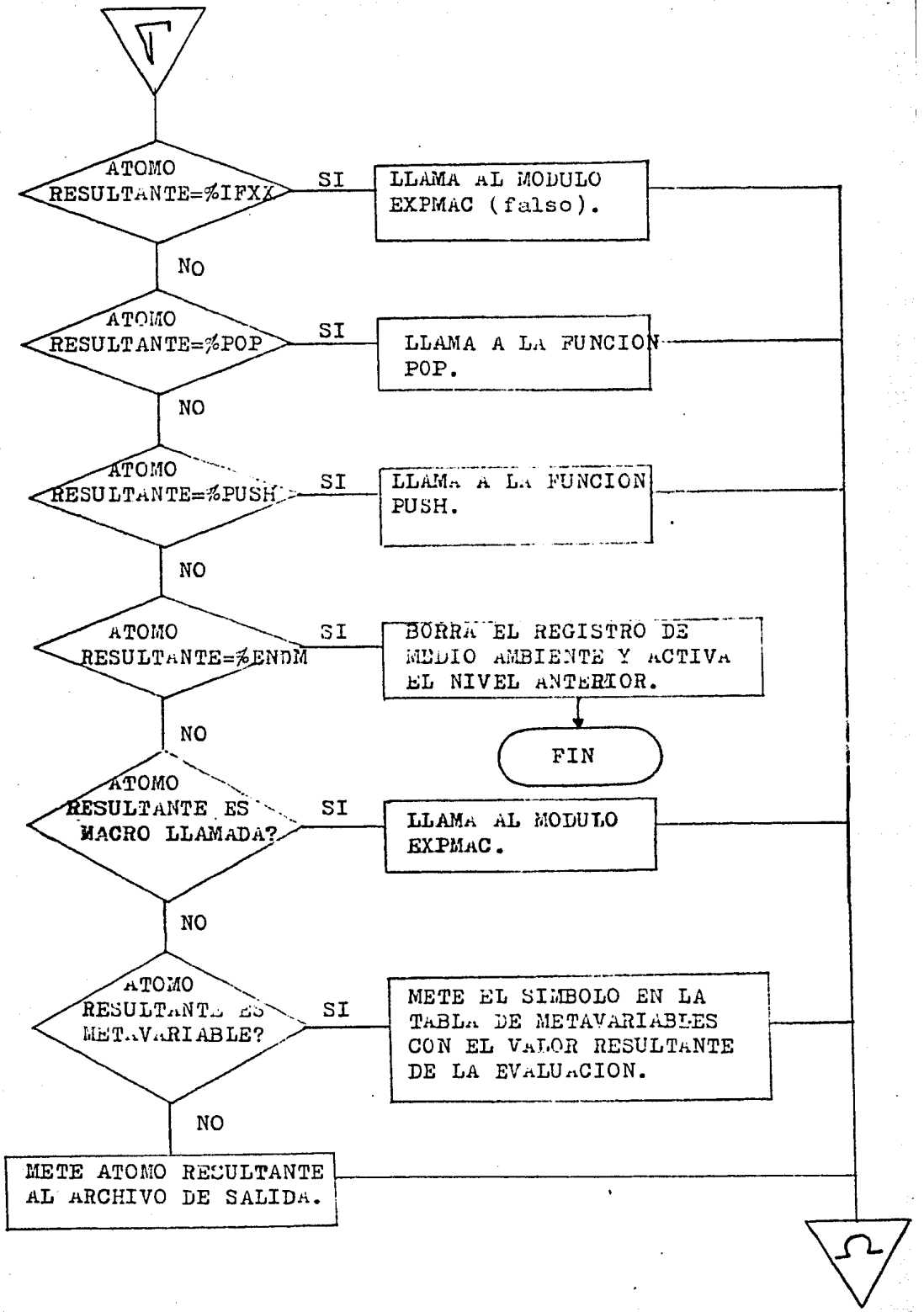

α
escribe directorio de macros, es decir, todos los macros definidos por el usuario.
escribe todos los simbolos definidos por el usuario.
FIN

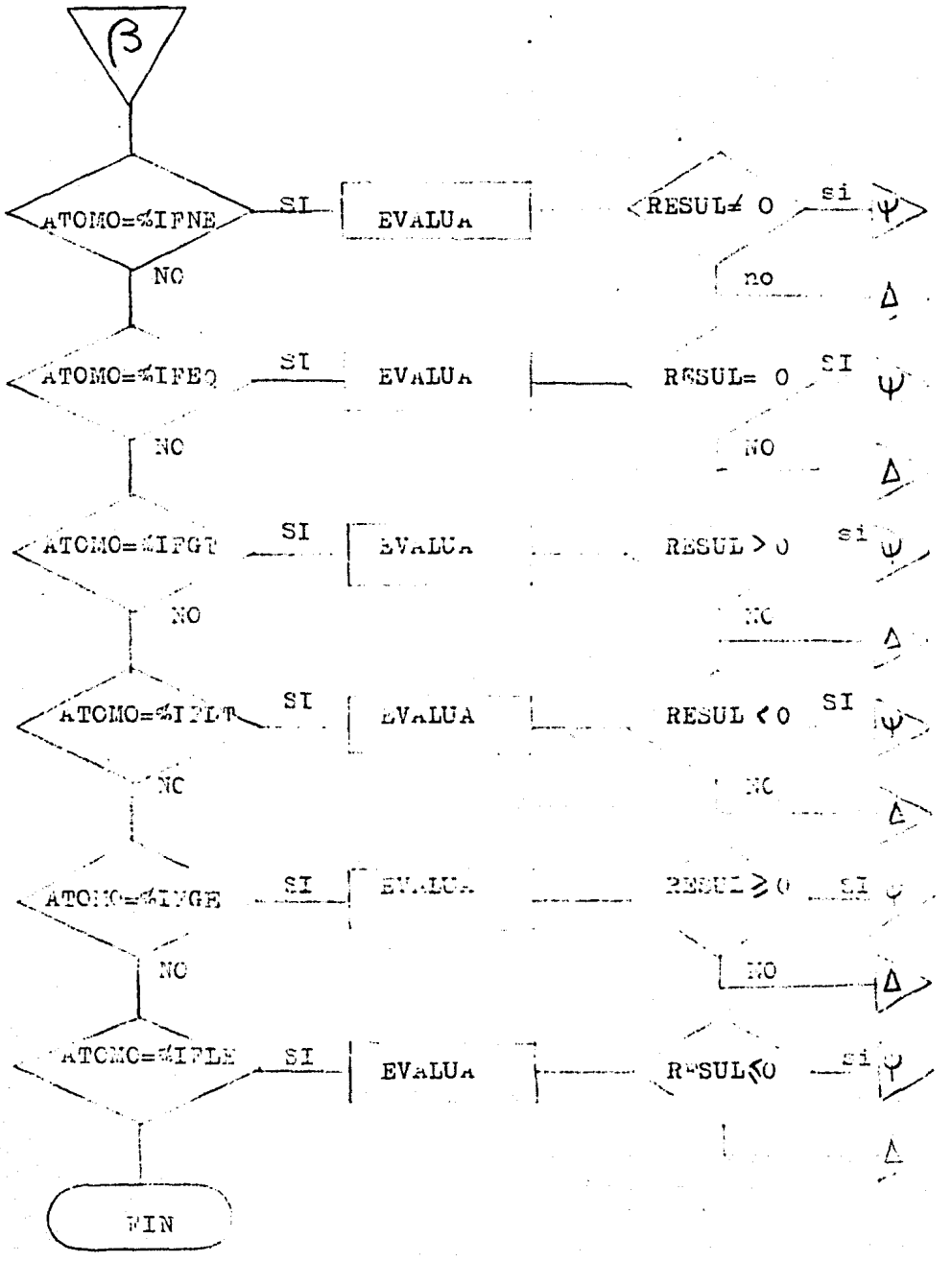
MODULO DEFMAC

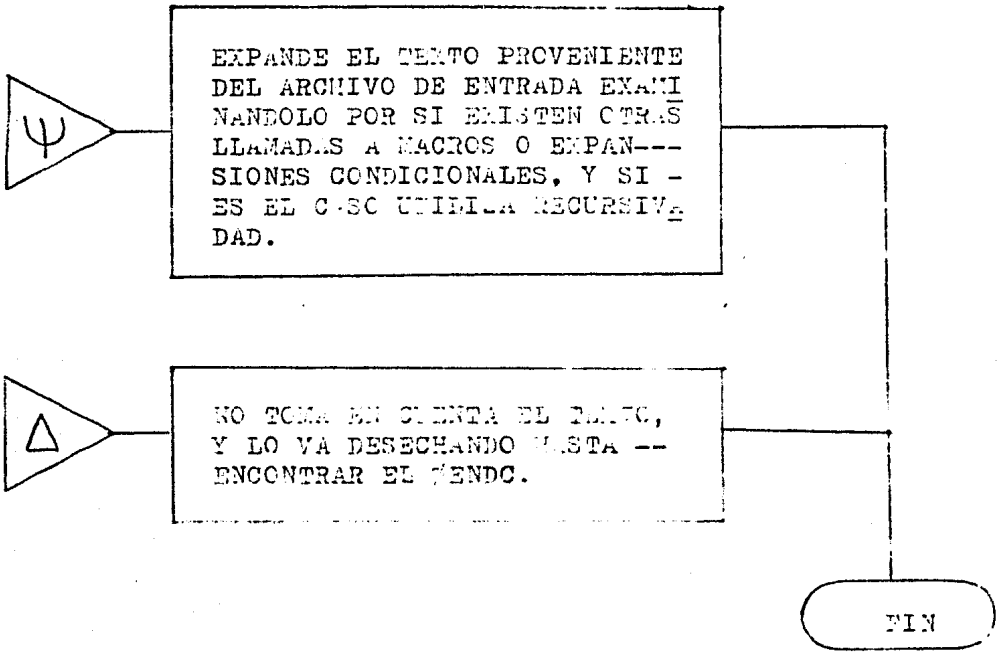


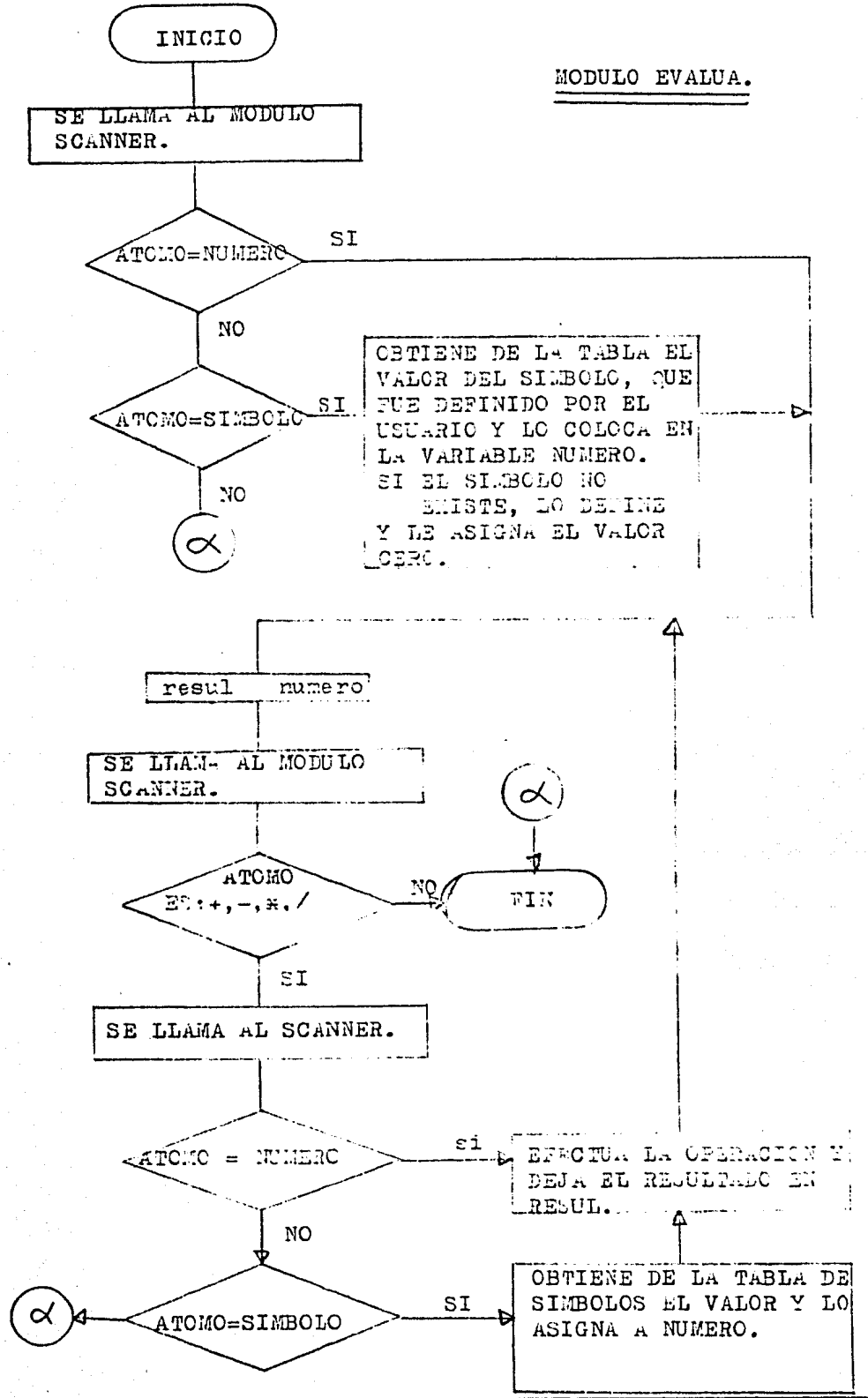
MODULO EXPMAC




ATOMO RESULTANTE=%IFXX
SI
LLAMA AL MODULO EXPMAC (falso).
NO
ATOMO RESULTANTE=%POP
SI
LLAMA A LA FUNCION POP.
NO
ATOMO RESULTANTE=%PUSH
SI
LLAMA A LA FUNCION PUSH.
NO
ATOMO RESULTANTE=%ENDM
SI
BORRA EL REGISTRO DE MEDIO AMBIENTE Y ACTIVA EL NIVEL ANTERIOR.
FIN
NO
ATOMO RESULTANTE ES MACRO LLAMADA?
SI
LLAMA AL MODULO EXPMAC.
NO
ATOMO RESULTANTE ES METAVARIABLE?
SI
METE EL SIMBOLO EN LA TABLA DE METAVARIABLES CON EL VALOR RESULTANTE DE LA EVALUACION.
NO
METE ATOMO RESULTANTE AL ARCHIVO DE SALIDA.


β
ATOMO=%IFNE
SI
EVALUA
RESUL≠ 0
si
Ψ
NO
no
Δ
ATOMO=%IFEQ
SI
EVALUA
RESUL= 0
SI
Ψ
NO
NO
Δ
ATOMO=%IFGT
SI
EVALUA
RESUL > 0
si
Ψ
NO
NO
Δ
ATOMO=%IFLT
SI
EVALUA
RESUL < 0
SI
Ψ
NO
NO
Δ
ATOMO=%IFGE
SI
EVALUA
RESUL ≥ 0
SI
Ψ
NO
NO
Δ
ATOMO=%IFLE
SI
EVALUA
RESUL ≤ 0
si
Ψ
Δ
FIN


Ψ
EXPANDE EL TEXTO PROVENIENTE DEL ARCHIVO DE ENTRADA EXAMINANDOLO POR SI EXISTEN OTRAS LLAMADAS A MACROS O EXPANSIONES CONDICIONALES, Y SI ES EL CASO UTILIZA RECURSIVIDAD.
Δ
NO TOMA EN CUENTA EL TEXTO, Y LO VA DESECHANDO HASTA ENCONTRAR EL FENDC.
FIN


INICIO
MODULO EVALUA.
SE LLAMA AL MODULO SCANNER.
ATOMO=NUMERO
SI
NO
ATOMO=SIMBOLO
SI
NO
OBTIENE DE LA TABLA EL VALOR DEL SIMBOLO, QUE FUE DEFINIDO POR EL USUARIO Y LO COLOCA EN LA VARIABLE NUMERO. SI EL SIMBOLO NO EXISTE, LO DEFINE Y LE ASIGNA EL VALOR CERO.
resul numero
SE LLAMA AL MODULO SCANNER.
ATOMO
NO
FIN
SI
SE LLAMA AL SCANNER.
ATOMO = NUMERO
si
EFECTUA LA OPERACION Y DEJA EL RESULTADO EN RESUL.
NO
ATOMO=SIMBOLO
SI
OBTIENE DE LA TABLA DE SIMBOLOS EL VALOR Y LO ASIGNA A NUMERO.

```
0               (*$E+*)
1               PROGRAM MAIN(TTY);
2       (*
3
4
5
6       PROGRAMA: MACRO PROCESADOR DE PROPOSITO GENERAL
7
8       AUTOR:    GRACIANO CRUZ ALMANZA
9
10      FECHA:    CIUDAD UNIVERSITARIA, A 30 DE ABRIL DE 1982.
11
12      ULT-MODIF:3 DE JULIO DE 1983
13
14
15
16      DESCRIPCION:
17                 ESTE PROGRAMA ES UN MACRO PROCESADOR DE PROPOSITO
18              GENERAL, EL CUAL TRABAJA COMO PREPROCESADOR. PERMITE
19              AL USUARIO TENER BIBLIOTECAS MACRO, DE TAL MANERA --
20              QUE SE DE POR LA TERMINAL EL NOMBRE DEL ARCHIVO QUE
21              CONTIENE LAS MACRO DEFINICIONES.
22              ESTE PROCESADOR TRABAJA EN MODO WARNNING, DANDO AL -
23              USUARIO LA FACILIDAD PARA CAMBIAR LOS CARACTERES ---
24              SI ASI LO DESEA. MPPG TIENE DEFINIDOS CARACTERES ---
25              WARNING DE DEFAULT ( # , % ).
26              ESTE PROCESADOR MANEJA EL TEXTO EN FORMA DE ATOMOS -
27              Y SUS PRINCIPALES CARACTERISTICAS SON LA DE PERMITIR
28              DEFINIR MACROS DENTRO DE MACROS, PERMITE ANIDACION -
29              DE MACROS, RECURSION, TIENE FUNCIONES PREDEFINIDAS Y
30              PROCESO INTERACTIVO.
31
32
33
34      ENTRADA:
35              A) RESPONDER SI (Y) O NO (N) AL PREGUNTAR MPPG.
36                 SI LA RESPUESTA FUE AFIRMATIVA, HAY QUE DAR LOS -
37                 NUEVOS CARACTERES WARNING ACORDE A COMO LOS PIDA
38                 MPPG.
39              B) RESPONDER SI (Y) O NO (N) AL PREGUNTAR MPPG SOBRE
40                 LAS BIBLIOTECAS MACRO.
41                 SI LA RESPUESTA FUE AFIRMATIVA, HAY QUE DAR EL --
42                 NOMBRE DEL ARCHIVO EL CUAL CONTIENE LAS MACRO-DE_
43                 FINICIONES.
```

```
45                  PIDA.
46
47
48
49          SALIDA:
50                  A) POR LA TERMINAL IMPRIMIRA LOS NOMBRES DE LOS MA__
51                     CROS DEFINIDOS POR EL USUARIO JUNTO CON SU APUN__
52                     TADOR AL ARCHIVO DE MACROS, DICHO ARCHIVO SE LLA_
53                     MA: AUXILIAR.TMP.
54                  B) POR LA TERMINAL IMPRIMIRA TODAS LAS METAVARIABLES
55                     CON SUS VALORES FINALES RESPECTIVOS.
56                  C) CONSTRUYE UN ARCHIVO LLAMADO SALIDA.TMP, EL CUAL
57                     CONTIENE EL TEXTO FUENTE JUNTO CON SUS MACRO YA -
58                     EXPANDIDOS. EVIDENTEMENTE EN ESTE ARCHIVO NO HAY
59                     MACRO DEFINICIONES YA.
60                  D) CONSTRUYE UN ARCHIVO LLAMADO AUXILIAR.TMP, EL CUAL
61                     CONTIENE LOS CUERPOS DE LOS MACROS DEFINIDOS POR -
62                     EL USUARIO JUNTO CON SUS 'ENDM'.
63
64
65
66          *)
67                  LABEL   100;
68                  CONST   MAXLON = 15;
69          (* *)
70                  TYPE    ID      = PACKED ARRAY[1..MAXLON] OF CHAR;
71          (* *)
72                          REGCRO = RECORD
73                          PARFOR : ID;
74                          SIM    : 0..3
75                          END;
76          (* *)
77                          REGUNO = RECORD
78                          BUPAAC : PACKED ARRAY[1..MAXLON] OF REGCRO;
79                          RETURN : INTEGER
80                          END;
81          (* *)
82                          REGDOS = RECORD
83                          NOMBRE : ID;
84                          LIGARC : INTEGER
85                          END;
86          (* *)
87                          REGTRE = RECORD
88                          SYMBOL : ID;
89                          VALOR  : INTEGER
```

```
90                      END;
91      (* *)
92              VAR     REGMA   : PACKED ARRAY[1..MAXLON] OF REGUNO;
93                      DIRMAC  : PACKED ARRAY[1..MAXLON] OF REGDOS;
94                      TABSYM  : PACKED ARRAY[1..MAXLON] OF REGTRE;
95                      PARAFO  : PACKED ARRAY[1..MAXLON] OF ID;
96                      NOM     : ID;
97                      STACK    : ARRAY[1..MAXLON] OF INTEGER;
98                      SP,
99                      APRGMA ,
100                     APDIR  ,
101                     APTASY ,
102                     APPAFO ,
103                     APFILE ,
104                     CONNOM ,
105                     RESUL  ,
106                     SPACES : INTEGER;
107                     APBUAC : ARRAY[1..MAXLON] OF INTEGER;
108                     E,F,G,H: FILE OF CHAR;
109                     G1      : FILE OF CHAR;
110                     CC,
111                     CH,CH1 : BYTE;
112                     SIMBOL : 0..3;
113                     I,J,K,
114                     NUMERO : INTEGER;
115                     IDENTO,
116                     IDENTI : ID;
117                     FLAG5,
118                     FLAG4,
119                     FLAG3,
120                     FLAG2,
121                     FLAG1,
122                     FLAG0,
123                     FLAG    : BOOLEAN;
124     (*************************************************************************)
125     PROCEDURE CLOSEF(  VAR F : TEXT );
126     EXTERN;
127     (*************************************************************************)
128     PROCEDURE ERROR( W : INTEGER );
129     (* ESTE PROCEDURE ESCRIBIRA EN LA TERMINAL
130        EL ERROR EN QUE EL USUARIO INCURRIO. *)
131     BEGIN
132     WRITELN(TTY,' ERROR ---> ',W);
133     END;
134     (*************************************************************************)
```

```
135     PROCEDURE PUSH;
136     BEGIN
137     IF (SP>MAXLON) THEN ERROR(18);
138     STACK[SP]:=TABSYM[1].VALOR;
139     SP:=SP+1;
140     END;
141     (*************************************************************************)
142     PROCEDURE POP;
143     BEGIN
144     IF (SP=1) THEN ERROR(20);
145     SP:=SP-1;
146     TABSYM[2].VALOR:=STACK[SP];
147     END;
148     (*************************************************************************)
149     PROCEDURE  ESC;
150     VAR      AUX : INTEGER;
151     BEGIN
152     AUX:=TABSYM[2].VALOR;
153     WRITE(H,AUX:5);
154     END;
155     (*************************************************************************)
156     PROCEDURE COPIA( I : INTEGER );
157     (*       ESTE MODULO EFECTUA COPIAS ENTRE ARCHIVOS
158              DE LA SIGUIENTE FORMA:
159              SI I=1 ENT. COPIA AUXILIAR.TMP A SOL.TMP
160              SI I#1 ENT. COPIA SOL.TMP A AUXILIAR.TMP     *)
161     VAR      CAR : CHAR;
162     BEGIN
163     IF ( I = 1 ) THEN
164              BEGIN
165              RESET(G,'AUXILIAR.TMP');
166              IF (IORESULT(G)<0) THEN ERROR(56);
167              REWRITE(G1,'SOL.TMP');
168              IF (IORESULT(G1)<0) THEN ERROR(58);
169              END
170     ELSE
171              BEGIN
172              RESET(G,'SOL.TMP');
173              IF (IORESULT(G)<0) THEN ERROR(58);
174              REWRITE(G1,'AUXILIAR.TMP');
175              IF (IORESULT(G1)<0) THEN ERROR(56);
176              END;
177     WHILE NOT(EOF(G)) DO
178              BEGIN
179              WHILE NOT(EOLN(G)) DO
```

```
180                     BEGIN
181                     READ(G,CAR);
182                     WRITE(G1,CAR);
183                     END;
184             READLN(G);
185             WRITELN(G1);
186             END;
187     END;
188     (*************************************************************************)
189     PROCEDURE ESCDIR;
190     (*      ESTE MODULO ESCRIBE EL DIRECTORIO DE MACROS,
191             ES DECIR, TODOS LOS MACRO QUE FUERON DEFINI_
192             DOS POR EL USUARIO.                             *)
193     VAR     K,L : INTEGER;
194     BEGIN
195     WRITELN(TTY);
196     WRITELN(TTY);
197     WRITELN(TTY,' EL DIRECTORIO DE MACROS ES:');
198     FOR K:=1 TO APDIR DO
199             BEGIN
200             WRITE(TTY,'EL MACRO SE LLAMA ');
201             FOR L:=1 TO MAXLON DO WRITE(TTY,DIRMAC[K].NOMBRE[L]);
202             WRITELN(TTY,' Y SU AP. AL ARCHIVO ES:', DIRMAC[K].LIGARC);
203             END;
204     END; (* END DEL PROCEDURE *)
205     (*************************************************************************)
206     PROCEDURE ESCTAB;
207     (*      ESTE PROCEDURE ESCRIBE LA TABLA DE SIMBOLOS
208             DEFINIDOS POR EL USUARIO                        *)
209     VAR     K,L : INTEGER;
210     BEGIN
211     WRITELN(TTY);
212     WRITELN(TTY);
213     WRITELN(TTY);
214     WRITELN(TTY,' TABLA DE SIMBOLOS DEF. ');
215     FOR K:=1 TO APTASY DO
216             BEGIN
217             WRITE(TTY,' EL SIMBOLO ES:');
218             FOR L:=1 TO MAXLON DO WRITE(TTY,TABSYM[K].SYMBOL[L]);
219             WRITELN(TTY,' Y SU VALOR ES:',TABSYM[K].VALOR);
220             END;
221     END;
222     (*************************************************************************)
223             PROCEDURE SCANNER( VAR  Y:TEXT);
224     (*      SCANNER:
```

```
225             ESTE MODULO ES UN SCANNER DEL CUAL SE OBTIENE COMO SALIDA
226             UN TOKEN EN LA VARIABLE IDENTI SI ES ALFABETICO,    JUNTO
227             CON SU LONGITUD EN LA VARIABLE K. SI EL TOKEN ES NUMERICO
228             ENTONCES EL VALOR RETORNADO ESTARA EN LA VARIABLE NUMERO.
229             EN LA VARIABLE SIMBOL DARA UN NUMERO ENTERO QUE CORRESPON
230             DE A:
231             0       SI ES CARACTER ESPECIAL.
232             1       SI ES UN NUMERO.
233             2       SI ES UN IDENTIFICADOR.
234             3       SI ES UNA PSEUDO-INSTRUCCION.                    *)
235     (***************)
236     PROCEDURE LEECHR;
237     BEGIN
238     IF FLAG2 THEN CH:=' ' ELSE READ(Y,CH);
239     IF EOLN(Y) THEN
240             BEGIN
241             FLAG1:=TRUE;
242             FLAG3:=TRUE;
243             END;
244     IF EOF(E) THEN FLAG0:=TRUE;
245     IF EOF(F) THEN FLAG2:=TRUE;
246     END;
247     (***************)
248     PROCEDURE CONSIM;
249     BEGIN
250     K:=0;
251     REPEAT
252             IF K < MAXLON THEN
253                     BEGIN
254                     K:=K+1;
255                     IF (K>MAXLON) THEN ERROR(22);
256                     IDENTIC[K]:=CH;
257                     END;
258             LEECHR;
259     UNTIL ( (NOT ( CH IN ['A'..'Z','0'..'9'] )) OR (EOLN(Y) ) );
260     IF CH IN ['A'..'Z','0'..'9'] THEN
261             BEGIN
262             K:=K + 1;
263             IF (K>MAXLON) THEN ERROR(22);
264             IDENTIC[K]:=CH;
265             LEECHR;
266             END;
267     END;
268     (***************)
269     BEGIN  (* INICIO DEL PROCEDURE SCANNER, SALTA BLANCOS*)
```

```
270     FOR I:=1 TO MAXLON DO IDENTIC I]:=' ';
271     SPACES:=0;
272     IF ( FLAG3 ) THEN
273             BEGIN
274             SPACES:=SPACES-1;
275             FLAG3:=FALSE;
276             END;
277     WHILE ( CH = ' ' ) AND ( NOT(FLAG2) ) DO
278             BEGIN
279             LEECHR;
280             SPACES:=SPACES+1;
281             END;
282     IF (FLAG5) THEN
283             BEGIN
284             SPACES:=SPACES-1;
285             FLAG5:=FALSE;
286             END;
287     IF CH = CC THEN
288             BEGIN
289             CH1:=CH;
290             LEECHR;
291             IF CH IN C'A'..'Z'] THEN
292                     BEGIN
293                     SIMBOL:=3;
294                     CONSIM;
295                     END     ELSE
296                     BEGIN
297                     SIMBOL:=0;
298                     IDENTIC1]:=CH1;
299                     END
300             END     ELSE
301     IF CH IN C'A'..'Z'] THEN
302             BEGIN
303             SIMBOL:=2;
304             CONSIM;
305             END     ELSE
306     IF CH IN C'0'..'9'] THEN
307             BEGIN
308             SIMBOL:=1;
309             NUMERO:=0;
310             K:=0;
311             REPEAT
312                     K:=K+1;
313                     NUMERO:=10*NUMERO+ORD(CH)-ORD('0');
314                     LEECHR;
```

```
315             UNTIL ( (NOT ( CH IN ['0'..'9'] )) OR (EOLN(Y) ) )
316     (* *)
317             IF CH IN ['0'..'9'] THEN
318                     BEGIN
319                     NUMERO:=10*NUMERO+ORD(CH)-ORD('0');
320                     CH:=' ';
321                     END;
322             IF (NUMERO<0) THEN ERROR(24);
323             END     ELSE
324             BEGIN  (* CARACTER ESPECIAL *)
325             SIMBOL := 0 ;
326             IDENTI[1]:=CH;
327             LEECHR;
328             END;
329     END; (* SCANNER *)
330     (*************************************************************************)
331     PROCEDURE BUSSYM(   VAR Y,Z : INTEGER );
332     (* ESTE PROCEDURE VERIFICA SI EL IDENTIFICADOR
333        ALMACENADO EN IDENTI ES UN SIMBOLO DEFINIDO
334        POR EL USUARIO.
335        SI EL SIM. YA FUE DEFINIDO, ENTONCES:
336        A) DA EN Y EL VALOR DE DICHA METAVARIABLE.
337        B) DA EN X EL LUGAR EN LA TABLA DE SIMBOLOS
338        SI EL SIMBOLO NO HA SIDO DEFINIDO, ENTONCES:
339        A) DEFINE EL SIMBOLO( LO METE A LA TABLA ).
340        B) ASIGNA EL VALOR CERO A DICHO SIMBOLO.
341        C) DA EN Z EL LUGAR EN LA TABLA DE SIMBOLOS.*)
342     VAR     I       : INTEGER;
343             FLAG    : BOOLEAN;
344     BEGIN
345     FLAG:=FALSE;
346     Y:=0;
347     FOR I:=1 TO APTASY DO
348             BEGIN
349             IF (TABSYM[I].SYMBOL = IDENTI ) THEN
350                     BEGIN
351                     FLAG:=TRUE;
352                     Y:=TABSYM[I].VALOR;
353                     Z:=I;
354                     END;
355             END;
356      IF NOT(FLAG) THEN
357             BEGIN
358             APTASY:=APTASY+1;
359             TABSYM[APTASY].SYMBOL:= IDENTI ;
```

```
360             TABSYM[APTASY].VALOR:=0;
361             Z:=APTASY;
362             END;
363     END;
364     (*************************************************************************)
365     PROCEDURE EVALUA( VAR X: TEXT);
366     (* ESTE PROCEDURE EVALUA UNA EXPRESION DE IZQ.
367        A DERECHA SIN REGLAS DE PRIORIDAD. EN LA --
368        EXPRESION PUEDE HABER METAVARIABLES O CONS_
369        TANTES.
370        LOS OPERADORES PERMITODOS SON SUMA, RESTA,
371        MUNTIPLICACION Y DIVISION. ES EVIDENTE QUE
372        LA ARITMETICA USADA ES ENTERA DEBIDO A LA
373        CARACTERISTICA DE USO DE DICHAS METAVARI_
374        ABLES EN MPPG.
375        COMO ENTRADA SE AL PROCEDURE SE DA EL AR_
376        CHIVO DE DONDE OBTENDRA LA EXPRESION Y DA
377        COMO SALIDA EL RESULTADO DE LA EVALUACION
378        EN LA VARIABLE GLOBAL RESUL.
379        SI EXISTE UN ERROR EN LA EXPRESION, LA --
380        EVALUACION SE DETENDRA Y TOMARA EL VALOR
381        QUE HASTA ESE MOMENTO TENGA RESUL.      *)
382     VAR     I : INTEGER;
383             FLAG  : BOOLEAN;
384             CARESP : CHAR;
385     BEGIN
386     RESUL:=0;
387     FLAG:=FALSE;
388     SCANNER(X);
389     IF ( (SIMBOL=1) OR (SIMBOL=3) ) THEN
390             BEGIN
391             IF (SIMBOL=3) THEN BUSSYM(NUMERO,I);
392             RESUL:=NUMERO;
393             REPEAT
394                     SCANNER(X);
395                     IF ((IDENTIC1]#'+')AND(IDENTIC1]#'-')AND
396                             (IDENTIC1]#'*')AND(IDENTIC1]#'/'))
397                             THEN FLAG:=TRUE;
398                     IF NOT(FLAG) THEN
399                             BEGIN
400                             CARESP:=IDENTIC1];
401                             SCANNER(X);
402                             IF ((SIMBOL#1)AND(SIMBOL#3)) THEN FLAG:=TRUE
403                                     ELSE
404                                     BEGIN
```

```
405                                     IF (SIMBOL=3) THEN BUSSYM(NUMERO,I);
406                                     CASE CARESP OF
407                                             '+':RESUL:=RESUL+NUMERO;
408                                             '-':RESUL:=RESUL-NUMERO;
409                                             '*':RESUL:=RESUL*NUMERO;
410                                             '/':RESUL:=RESUL DIV NUMERO
411                                             END; (* END DEL CASE *)
412                                     END;
413                             END;
414                     FLAG1:=FALSE;
415             UNTIL FLAG;
416             END;
417     END;
418     (*************************************************************************)
419     PROCEDURE ESMAC( VAR Z : BOOLEAN );
420     (* ESTE PROCEDURE VE SI IDENTI ES UNA MACRO-LLAMADA,
421        DICHA BUSQUEDA SE EFECTUA EN EL DIRECTORIO DE MA_
422        CROS.                                              *)
423     VAR     I       : INTEGER;
424     BEGIN
425     Z:=FALSE;
426     FOR I:=1 TO APDIR DO
427             BEGIN
428             IF IDENTI = DIRMAC[I].NOMBRE THEN Z:=TRUE;
429             END;
430     END;
431     (*************************************************************************)
432     PROCEDURE GETPAR( VAR ZZZ:TEXT );
433     (* ESTE PROCEDURE ES EL ENCARGADO DEOBTENER LOS
434        PARAMETROS FORMALES, LOS CUALES SON ALMACENADOS
435        EN EL ARREGLO PARFOR[APPAFO].
436        LOS PARAMETROS SON ALMACENADOS PARA REALIZAR --
437        POSTERIORMENTE LA SUSTITUCION DE DICHOS PARAME_
438        TROS, POR PARAMETROS WARNING.
439        EL PARAMETRO DEL PROCEDURE ES LA VARIABLE ASIG_
440        NADA AL ARCHIVO DE DONDE SE OBTENDRAN LOS PARA_
441        METROS FORMALES.                                 *)
442     VAR     ERROR : BOOLEAN;
443     BEGIN
444     APPAFO:=0;
445     ERROR:=FALSE;
446     SCANNER(ZZZ);
447     IF ( SIMBOL # 0 ) OR ( IDENTI[1] # '(' ) THEN ERROR:=TRUE
448     ELSE
449         REPEAT
```

```
450             BEGIN
451             SCANNER(ZZZ);
452             IF (SIMBOL#0)OR(IDENTI[1]#')') THEN
453             IF (SIMBOL#2) THEN ERROR:=TRUE
454             ELSE
455                 BEGIN
456                 APPAFO:=APPAFO+1;
457                 PARAFO[APPAFO]:=IDENTI;
458                 SCANNER(ZZZ);
459                 END;
460             END UNTIL ( (IDENTI[1]#',') OR (SIMBOL#0) );
461     IF (SIMBOL#0) AND (IDENTI[1]#')') THEN ERROR:=TRUE;
462     WHILE ERROR DO
463             BEGIN
464             SCANNER(ZZZ);
465             IF IDENTI[1]=')' THEN ERROR:=FALSE;
466             END;
467     FLAG3:=TRUE;
468     END;
469     (*************************************************************************)
470     PROCEDURE DEFMAC( VAR X,Y:TEXT );
471     VAR     I,NIVEL : INTEGER;
472     (***************)
473     BEGIN
474     NIVEL:=1;
475     APDIR:=APDIR + 1 ;
476     SCANNER(X);
477     IF(SIMBOL <> 2) THEN ERROR(10)
478             ELSE
479             BEGIN
480             DIRMAC[APDIR].NOMBRE:=IDENTI;(*TAL VEZ LA ASIGNACION SEA DIRECTA*)
481             DIRMAC[APDIR].LIGARC:=APFILE;
482             GETPAR(X);
483             FLAG1:=FALSE;
484             WHILE ( NIVEL > 0 ) DO
485                     BEGIN
486                     SCANNER (X);
487                     FLAG:=FALSE;
488                     IF (IDENTI='MACRO          ') THEN NIVEL:=NIVEL + 1;
489                     FOR I:=1 TO APPAFO DO
490                             BEGIN
491                             IF IDENTI=PARAFO[I] THEN
492                                             BEGIN
493                                             NUMERO:=I;
494                                             FLAG:=TRUE;
```

```
495                                         END;
496                             END;
497                     IF ( FLAG ) THEN
498                                         BEGIN
499                                         FOR I:=1 TO SPACES DO WRITE(Y,' ');
500                                         WRITE(Y,'\',NUMERO);
501                                         END
502                                         ELSE
503                                         BEGIN
504                                         FOR I:=1 TO SPACES DO WRITE(Y,' ');
505                                         CASE SIMBOL OF
506                                         0: WRITE(Y,IDENTIC1]);
507                                         1: WRITE(Y,NUMERO:K);
508                                         2: FOR I:=1 TO K DO WRITE(Y,IDENTIC1]);
509                                         3: BEGIN
510                                            WRITE(Y,CC);
511                                            FOR I:=1 TO K DO WRITE(Y,IDENTIC1]);
512                                            END
513                                         END; (* CASE *)
514                                         END;
515                     IF FLAG1 THEN
516                             BEGIN
517                             APFILE:=APFILE + 1;
518                             WRITELN(Y);
519                             FLAG1:=FALSE;
520                             END;
521                     IF ((IDENTI='ENDM           ') AND (NIVEL <> 0 )) THEN
522                                             NIVEL:=NIVEL-1;
523                     END;
524             END;
525     END;
526     (*************************************************************************)
527     PROCEDURE GETACT( VAR ZZZ : TEXT );
528     VAR     ERROR   : BOOLEAN;
529             P       : INTEGER;
530     BEGIN
531     APBUACCAPRGMA]:=0;
532     ERROR:=FALSE;
533     SCANNER(ZZZ);
534     IF ( SIMBOL#0) OR (IDENTIC1]#'(') THEN ERROR:=TRUE
535     ELSE
536         REPEAT
537             BEGIN
538             SCANNER(ZZZ);
539             IF (SIMBOL#0)OR(IDENTIC1]#')') THEN
```

```
540                     BEGIN
541                     IF (IDENTIC1]='C')  THEN
542                             BEGIN
543                             P:=1;
544                             CH:=' ';
545                             WHILE ( CH # ']' ) DO
546                                     BEGIN
547                                     READ(ZZZ,CH);
548                                     IDENTIC P]:=CH;
549                                     P:=P+1;
550                                     END;
551                             IDENTIC P-1]:=' ';
552                             CH:=' ';
553                             SIMBOL:=2;
554                             END;
555                     APBUACCAPRGMA]:=APBUACCAPRGMA] + 1 ;
556                     CASE SIMBOL OF
557                             0:BEGIN
558                               REGMACAPRGMA].BUPAACC APBUACCAPRGMA] ].PARFOR:=IDENTI;
559                               REGMACAPRGMA].BUPAACC APBUACCAPRGMA] ].SIM:=0;
560                               END;
561                             1:BEGIN
562                               WHILE NUMERO > 0 DO
563                                     BEGIN
564                                     I:=MAXLON;
565                                     REPEAT
566                                             BEGIN
567                                             IDENTIC I]:=IDENTIC I-1];
568                                             I:=I-1;
569                                             END UNTIL (I=1);
570                                     IDENTIC1]:=CHR((NUMERO MOD 10) + ORD('0'));
571                                     NUMERO:=NUMERO DIV 10;
572                                     END;
573                               REGMACAPRGMA].BUPAACC APBUACCAPRGMA] ].PARFOR:=IDENTI;
574                               REGMACAPRGMA].BUPAACC APBUACCAPRGMA] ].SIM:=2;
575                               END;
576                             2:BEGIN
577                               REGMACAPRGMA].BUPAACC APBUACCAPRGMA] ].PARFOR:=IDENTI;
578                               REGMACAPRGMA].BUPAACC APBUACCAPRGMA] ].SIM:=2;
579                               END;
580                             3:BEGIN
581                               REGMACAPRGMA].BUPAACC APBUACCAPRGMA] ].PARFOR:=IDENTI;
582                               REGMACAPRGMA].BUPAACC APBUACCAPRGMA] ].SIM:=3;
583                               END
584                     END; (* FIN DEL CASE *)
```

```
585                     SCANNER(ZZZ);
586                     END;
587             END UNTIL ((IDENTIC1]#',')OR(SIMBOL#0));
588     IF ( SIMBOL#0) AND(IDENTIC1]#')') THEN ERROR:=TRUE;
589     WHILE ERROR DO
590             BEGIN
591             SCANNER(ZZZ);
592             IF ( IDENTIC1]=')' ) THEN ERROR:=FALSE;
593             END;
594     FLAG3:=TRUE;
595     END;
596     (*************************************************************************)
597     PROCEDURE EXPMAC( VAR XX:TEXT; ZZ:BOOLEAN );
598     (*      ESTE PROCEDURE ES EL ENCARGADO DE EFECTUAR
599             CUALQUIER EXPANSION,  YA SEA POR ENCONTRAR
600             UNA MACRO-LLAMADA,  COMO POR HABER ENCON__
601             TRADO EXPANSION CONDICIONAL.
602             LA DESCRIPCION DE LOS PARAMETROS ES:
603             ZZ = TRUE => MACRO-LLAMADA.
604             ZZ = FALSE => EXPANSION CONDICIONAL Y USO
605                           DE XX COMO ARCHIVO FUENTE DE
606                           DONDE SE SACARA EL CUERPO DE
607                           LA EXPANSION CONDICIONAL. *)
608     LABEL   10;
609     VAR     FLAGN   : BOOLEAN;
610             BUPAAC  : PACKED ARRAYC1..MAXLONJ OF ID;
611             AUX,
612             APBUAC  : INTEGER;
613     (***************)
614     PROCEDURE GETMAC;
615     VAR     I,J : INTEGER;
616     BEGIN
617     I:=1;
618     RESET(G,'AUXILIAR.TMP');
619     LOOP
620             EXIT IF (( IDENTI = DIRMACCIJ.NOMBRE ) OR (I>APDIR) );
621             I:=I+1;
622             END; (* FIN DEL LOOP *)
623     IF (I<=APDIR) THEN FOR J:=1 TO DIRMACCIJ.LIGARC DO READLN(G)
624                   ELSE ERROR(50);
625     REGMACAPRGMAJ.RETURN:=DIRMACCIJ.LIGARC + 1;
626     FLAG1:=FALSE;
627     END;
628     (***************)
629     PROCEDURE EXPIFS( VAR XXX:TEXT; YYY:BOOLEAN );
```

```
630     VAR     N : INTEGER;
631     BEGIN
632     IF YYY THEN
633             BEGIN
634             WHILE ( IDENTI # 'ENDC            ') DO
635                     BEGIN
636                     (* FALTA PREGUNTAR POR LOS MACROS*)
637                     IF ( IDENTI='IFNE          ' ) OR
638                        ( IDENTI='IFEQ          ' ) OR
639                        ( IDENTI='IFGT          ' ) OR
640                        ( IDENTI='IFLT          ' ) OR
641                        ( IDENTI='IFGE          ' ) OR
642                        ( IDENTI='IFLE          ' ) THEN EXPMAC(XXX,FALSE)
643                     ELSE
644                     IF ((IDENTI='MACRO          ') AND
645                         (SIMBOL=3)) THEN
646                             BEGIN
647                             CLOSEF(G);
648                             COPIA(1);
649                             DEFMAC(XXX,G1);
650                             CLOSEF(G);
651                             CLOSEF(G1);
652                             COPIA(10);
653                             CLOSEF(G);
654                             CLOSEF(G1);
655                             END
656                     ELSE
657                     BEGIN
658                     FOR I:=1 TO SPACES DO WRITE(H,' ');
659                     CASE SIMBOL OF
660                             0: WRITE(H,IDENTIC1]);
661                             1: WRITE(H,NUMERO:K);
662                             2: FOR I:=1 TO K DO WRITE(H,IDENTIC1]);
663                             3: BEGIN
664                                REGMACAPRGMA].RETURN:=REGMACAPRGMA].RETURN+1;
665                                EXPMAC(XXX,TRUE);
666                                END
667                             END;
668                     END;
669                     IF FLAG1 THEN
670                             BEGIN
671                             WRITELN(H,'**');
672                             FLAG1:=FALSE;
673                             REGMACAPRGMA].RETURN:=REGMACAPRGMA].RETURN+1;
674                             END;
```

```
675                     SCANNER(XXX);
676                     END;
677             END
678             ELSE
679             BEGIN
680             N:=0;
681             WHILE ( (  IDENTI # 'ENDC            ') OR
682                     (N # 0 )     ) DO
683                     BEGIN
684                     FLAG1:=FALSE;
685                     IF (((  IDENTI='IFNE            ') OR
686                          (  IDENTI='IFEQ            ') OR
687                          (  IDENTI='IFLT            ') OR
688                          (  IDENTI='IFGE            ') OR
689                          (  IDENTI='IFLE            ') ) AND (SIMBOL=3)) THEN N:=N+1;
690                     IF ((SIMBOL=3) AND (IDENTI='ENDC            ')) THEN N:=N+1;
691                     SCANNER(XXX);
692                     END;
693             END;
694     FLAG1:=FALSE;
695     END;
696     (***************)
697     BEGIN
698     IF ZZ THEN
699     BEGIN
700     CLOSEF(G);
701     COPIA(1);
702     CLOSEF(G);
703     FLAGN:=FALSE;
704     APRGMA:=APRGMA+1;
705     IF (APRGMA>MAXLON) THEN ERROR(62);
706     GETMAC;
707     FLAG1:=FALSE;
708     REPEAT
709             BEGIN
710             SCANNER(G);
711             10:
712             IF IDENTI[1] = '\' THEN
713                                     BEGIN
714                                     AUX:=SPACES;
715                                     SCANNER(G);
716                                     SPACES:=AUX;
717                                     IDENTI:=REGMAC[APRGMA-1].BUPAAC[NUMERO].PARFOR;
718                                     K:=MAXLON;
719                                     WHILE (IDENTI[K]=' ') DO K:=K-1;
```

```
720                                 SIMBOL:=REGMACAPRGMA-1].BUFAACENUMERO].SIM;
721                                 END;
722             IF (SIMBOL=3) THEN
723                     BEGIN
724                     IF IDENTI = 'MACRO          ' THEN
725                                             BEGIN
726                                             DEFMAC(G,G1);
727                                             FLAG1:=FALSE;
728                                             END
729                     ELSE
730                     IF ( IDENTI='IFNE           ' ) OR
731                     ( IDENTI='IFEQ              ' ) OR
732                     ( IDENTI='IFGT              ' ) OR
733                     ( IDENTI='IFLT              ' ) OR
734                     ( IDENTI='IFGE              ' ) OR
735                     ( IDENTI='IFLE              ' ) THEN EXPMAC(G,FALSE)
736                     ELSE
737                     IF IDENTI = 'POP                ' THEN
738                                     BEGIN
739                                     POP;
740                                     FLAG1:=FALSE;
741                                     END
742                     ELSE
743                     IF IDENTI = 'PUSH           ' THEN
744                                             BEGIN
745                                             PUSH;
746                                             FLAG1:=FALSE;
747                                             END
748                     ELSE
749                     IF IDENTI = 'ESC            ' THEN ESC
750                     ELSE
751                     IF IDENTI = 'ENDM           ' THEN
752                                             BEGIN
753                                             FLAGN:=TRUE;
754                                             FLAG1:=FALSE;
755                                             END
756                     ELSE
757                     BEGIN
758                     FLAG:=FALSE;
759                     ESMAC(FLAG);
760                     IF FLAG THEN
761                             BEGIN
762                             FLAG1:=FALSE;
763                             IDENTO:=IDENTI;
764                             IDENTI:=IDENTO;
```

```
765                             EXPMAC(G,TRUE);
766                             END
767                             ELSE
768                             BEGIN
769                             BUSSYM(NUMERO,J);
770                             SCANNER(G);
771                             IF (IDENTIC1]='=')AND(SIMBOL=0) THEN
772                                     BEGIN
773                                     EVALUA(G);
774                                     TABSYMCJ].VALOR:=RESUL;
775                                     END;
776                             GOTO 10;
777                             END;
778                     END
779                     END
780             ELSE
781                     BEGIN
782                     FLAG:=FALSE;
783                     ESMAC(FLAG);
784                     IF FLAG THEN
785                             BEGIN
786                             FLAG1:=FALSE;
787                             IDENTO:=IDENTI;
788                             GETACT(G);
789                             IDENTI:=IDENTO;
790                             EXPMAC(G,TRUE);
791                             END
792                     ELSE
793                             BEGIN
794                             FOR I:=1 TO SPACES DO WRITE(H,' ');
795                             CASE SIMBOL OF
796                                     0:WRITE(H,IDENTIC1]);
797                                     1:WRITE(H,NUMERO:K);
798                                     2:FOR I:=1 TO K DO WRITE(H,IDENTICI])
799                                     END;
800                             END;
801                     END;
802             IF (FLAG1) THEN
803                     BEGIN
804                     WRITELN(H);
805                     FLAG1:=FALSE;
806                     REGMACAPRGMA].RETURN:=REGMACAPRGMA].RETURN + 1;
807                     END;
808             END UNTIL (FLAGN);
809     CLOSEF(G);
```

```
810     CLOSEF(G1);
811     COPIA(10);
812     CLOSEF(G);
813     CLOSEF(G1);
814     RESET(G,'AUXILIAR.TMP');
815     IF (IORESULT(G)<0) THEN ERROR(64);
816     FOR I:=1 TO REGMACAPRGMA-1].RETURN     DO READLN(G);
817     APRGMA:=APRGMA-1;
818     END
819     ELSE
820     BEGIN
821     IF ( IDENTI = 'IFNE            ') THEN
822             BEGIN
823             EVALUA(XX);
824             IF ( RESUL # 0 ) THEN EXPIFS(XX,TRUE)
825                     ELSE EXPIFS(XX,FALSE);
826             END
827     ELSE
828     IF ( IDENTI = 'IFEQ            ') THEN
829             BEGIN
830             EVALUA(XX);
831             IF ( RESUL = 0 ) THEN EXPIFS(XX,TRUE)
832                     ELSE EXPIFS(XX,FALSE);
833             END
834     ELSE
835     IF ( IDENTI ='IFGT             ' ) THEN
836             BEGIN
837             EVALUA(XX);
838             REGMACAPRGMA].RETURN:=REGMACAPRGMA].RETURN+1;
839             IF ( RESUL > 0 ) THEN EXPIFS(XX,TRUE)
840                     ELSE EXPIFS(XX,FALSE);
841             END
842     ELSE
843     IF ( IDENTI ='IFLT             ' )THEN
844             BEGIN
845             EVALUA(XX);
846             IF ( RESUL < 0 ) THEN EXPIFS(XX,TRUE)
847                     ELSE EXPIFS(XX,FALSE);
848             END
849     ELSE
850     IF ( IDENTI ='IFGE             ' ) THEN
851             BEGIN
852             EVALUA(XX);
853             IF ( RESUL >= 0 ) THEN EXPIFS(XX,TRUE)
854                     ELSE EXPIFS(XX,FALSE);
```

```
855             END
856     ELSE
857     IF ( IDENTI ='IFLE            ' )THEN
858             BEGIN
859             EVALUA(XX);
860             IF ( RESUL <= 0 ) THEN EXPIFS(XX,TRUE)
861                     ELSE EXPIFS(XX,FALSE);
862             END;
863     END;
864     END;
865     (*************************************************************************)
866             (* SE LEE SI HAY ARCHIVO DE DEF, DE MACROS *)
867             (* SE LEE EL NOMBRE DEL ARCHIVO FUENTE *)
868             (* SE LLENA EL ARCHIVO AUX. CON MACRODEFINICIONES *)
869     (*************************************************************************)
870     BEGIN
871     REWRITE(G,'AUXILIAR.TMP');
872     IF (IORESULT(G)<0) THEN ERROR(50);
873     REWRITE(H,'SALIDA.TMP');
874     IF (IORESULT(H)<0) THEN ERROR(52);
875     SP:=1;
876     APRGMA:=1;
877     APDIR:=0;
878     APFILE:=1;
879     APTASY:=0;
880     CH:=' ';
881     FLAG0:=FALSE;
882     FLAG1:=FALSE;
883     FLAG2:=FALSE;
884     FLAG3:=TRUE;
885     WRITELN(G,' ');
886     CC:='%';
887     WRITELN(TTY,'CAMBIO DE CARACTER DE ADVERTENCIA ?');
888     READ(TTY,CH);
889     WHILE NOT(EOLN(TTY)) DO READ(TTY,CH);
890     IF (CH='Y') THEN
891             BEGIN
892             WRITELN(TTY,'DAME EL CARACTER:');
893             READ(TTY,CC);
894             END;
895     WRITELN(TTY,'USARA SU PROPIA BIBLIOTECA?');
896     READ(TTY,CH);
897     WHILE NOT(EOLN(TTY)) DO READ(TTY,CH);
898     IF (CH='Y') THEN
899             BEGIN
```

```
900             WRITELN(TTY);
901             WRITELN(TTY,'DEME EL NOMBRE DE SU BIBLIOTECA');
902             I:=1;
903             READ(TTY,CH);
904             WHILE NOT(EOLN(TTY)) DO
905                     BEGIN
906                     READ(TTY,NOMCI]);
907                     I:=I+1;
908                     END;
909             RESET(E,NOM);
910             IF ( IORESULT(E) < 0 ) THEN ERROR(54);
911             REPEAT
912                     BEGIN
913                     SCANNER(E);
914                     IF (IDENTI='MACRO          ') THEN
915                                     BEGIN
916                                     CLOSEF(G);
917                                     COPIA(1);
918                                     DEFMAC(E,G1);
919                                     CLOSEF(G);
920                                     CLOSEF(G1);
921                                     COPIA(10);
922                                     CLOSEF(G);
923                                     CLOSEF(G1);
924                                     END
925                             ELSE
926                                     ERROR(10);
927             END UNTIL FLAG0;
928             CLOSEF(E);
929             FLAG0:=FALSE;
930             FLAG1:=FALSE;
931             FLAG2:=FALSE;
932             END;
933     WRITELN(TTY,'DEME EL NOMBRE DEL ARCHIVO FUENTE');
934     FOR I:=1 TO MAXLON DO NOMCI]:=' ';
935     I:=1;
936     READ(TTY,CH);
937     WHILE NOT(EOLN(TTY)) DO
938             BEGIN
939             READ(TTY,NOMCI]);
940             I:=I+1;
941             END;
942     RESET(F,NOM);
943     IF (IORESULT(F)<0) THEN ERROR(60);
944     CH:=' ';
```

```
945     FLAG3:=TRUE;
946     FLAG4:=FALSE;
947     REPEAT
948             BEGIN
949                     BEGIN
950                     SCANNER(F);
951                     100:IF SIMBOL=3 THEN
952                             BEGIN
953                             IF (IDENTI='DEFTTY          ') THEN
954                                             BEGIN
955                                             FLAG4:=TRUE;
956                                             IDENTI:='MACRO          ';
957                                             END;
958                             IF (IDENTI='MACRO           ') THEN
959                                             BEGIN
960                                             CLOSEF(G);
961                                             COPIA(1);
962                                             IF (FLAG4) THEN DEFMAC(TTY,G1)
963                                                     ELSE    DEFMAC(F,G1);
964                                             FLAG4:=FALSE;
965                                             CLOSEF(G);
966                                             CLOSEF(G1);
967                                             COPIA(10);
968                                             CLOSEF(G);
969                                             CLOSEF(G1);
970                                             END
971                             ELSE
972                             IF (IDENTI='POP             ') THEN
973                                             BEGIN
974                                             POP;
975                                             FLAG1:=FALSE;
976                                             END
977                             ELSE
978                             IF (IDENTI='PUSH            ') THEN
979                                             BEGIN
980                                             PUSH;
981                                             FLAG1:=FALSE;
982                                             END
983                             ELSE
984                             IF (IDENTI='ENDM            ') THEN ERROR(16)
985                             ELSE
986                             IF (IDENTI='IFNE            ') OR
987                                (IDENTI='IFEQ            ') OR
988                                (IDENTI='IFGT            ') OR
989                                (IDENTI='IFLT            ') OR
```

```
990                                 (IDENTI='IFGE           ') OR
991                                 (IDENTI='IFLE           ') THEN EXPMAC(F,FALSE)
992                             ELSE
993                             IF (IDENTI='EXPTTY          ') THEN
994                                         BEGIN
995                                         SCANNER(TTY);
996                                         IDENTO:=IDENTI;
997                                         GETACT(TTY);
998                                         IDENTI:=IDENTO;
999                                         EXPMAC(TTY,TRUE);
1000                                        END
1001                            ELSE
1002                            BEGIN
1003                            FLAG:=FALSE;
1004                            ESMAC(FLAG);
1005                            IF FLAG THEN
1006                                    BEGIN
1007                                    FLAG1:=FALSE;
1008                                    IDENTO:=IDENTI;
1009                                    GETACT(F);
1010                                    IDENTI:=IDENTO;
1011                                    EXPMAC(F,TRUE);
1012                                    END
1013                                    ELSE
1014                                    BEGIN
1015                                    BUSSYM(NUMERO,J);
1016                                    SCANNER(F);
1017                                    IF (IDENTIC1]='=')AND(SIMBOL=0) THEN
1018                                            BEGIN
1019                                            EVALUA(F);
1020                                            TABSYMCJ].VALOR:=RESUL;
1021                                            END;
1022                                    GOTO 100;
1023                                    END;
1024                            END;
1025                            END
1026                            ELSE
1027                            BEGIN
1028                            IF ((FLAG1)AND(SIMBOL=0)) THEN FLAG5:=TRUE;
1029                            FOR I:=1 TO SPACES DO WRITE(H,' ');
1030                            CASE SIMBOL OF
1031                            0: WRITE(H,IDENTIC1]);
1032                            1: WRITE(H,NUMERO:K); (*VER SI SE PUEDE*)
1033                            2: FOR I:=1 TO K DO WRITE(H,IDENTIC1])
1034                            END; (* END DEL CASE *)
```

```
1035                                END;
1036                        END;
1037                IF FLAG1 THEN
1038                        BEGIN
1039                        WRITELN(H);
1040                        FLAG1:=FALSE;
1041                        END;
1042                IF EOF(F) THEN FLAG2:=TRUE;
1043                END UNTIL (FLAG2);
1044        ESCDIR;
1045        ESCTAB;
1046        WRITELN(TTY,'EL AP. AL ARCHIVO =', APFILE);
1047        END.
>
```

4.DESCRIPCION DE LOS MENSAJES DE ERROR.

Los mensajes de error de MPPG tienen el siguiente formato:

ERROR ---- n

Donde n es un numero entero que representa el tipo de error, acorde a la siguiente lista.

10 - Identificador distinto de %MACRO en la biblioteca de Macros. Detectado en el modulo MAIN.

12 - Identificador no valido como nombre de Macro.

14 - Error en la lista de parametros formales. Error - detectado en el modulo GETPAR.

16 - %ENDM encontrado sin haber Macro definicion.

18 - Sobrecupo en la pila del usuario, valor perdido.

20 - Imposible sacar elemento de la pila del usuario, pila vacia.

22 - Identificador demasiado largo, error detectado en el modulo SCANNER.

24 - Constante demasiado larga, error detectado en el - modulo SCANNER.

50 - Falla en apertura del archivo auxiliar, error en el modulo MAIN.

52 - Falla en apertura del archivo salida, error en el modulo MAIN.

54 - Falla en apertura del archivo que fue dado como biblioteca, error detectado en el modulo MAIN.

56 - Falla en apertura del archivo auxiliar, error detectado en el modulo COPIA.

58 - Falla en apertura del archivo SOL, error en el modulo COPIA.

60 - Falla en apertura del archivo fuente dado por el usuario. Error en el modulo MAIN.

62 - Demasiados niveles de anidacion en Macro llamadas.

64 - Falla en apertura del archivo auxiliar. Error en el modulo EXPMAC.

```
>
>; EJEMPLO PARA EL USO DE WHILE EN FORTRAN.
>
>PIP TI:=WHILE.TXT
%ETI=1
%AUX=1
%MACRO WHILE( Y )
%ETI=%ETI+1
%AUX=%ETI
%PUSH
%ESC    CONTINUE
%ETI=%ETI+1
%AUX=%ETI
        IF (.NOT.( Y ) GO TO %ESC
%POP
%PUSH
%CHANO=%ETI
%ETI=%AUX
%PUSH
%ETI=%CHANO
%ENDM
%MACRO ENDW( )
%POP
        GO TO %ESC
%POP
%ESC    CONTINUE
%ENDM
C       COMMENTS
        I=KIL
        %WHILE( FLAG2 )
        J=II
        %WHILE(FLAG)
                CHANO=123+CHANO
                MEMOR=JJ+MEMOR
        %ENDW()
        K=JJ
        L=MIS
        %ENDW( )
        I=KIL
        END.
>
>; SE LLAMA A MPPG
>
>@A
>PIP *.TMP;*/DE
>RUN MPPG
CAMBIO DE CARACTER DE ADVERTENCIA ?
N
USARA SU PROPIA BIBLIOTECA?
N
DEME EL NOMBRE DEL ARCHIVO FUENTE
WHILE.TXT
 EL DIRECTORIO DE MACROS ES:
EL MACRO SE LLAMA WHILE          Y SU AP. AL ARCHIVO ES:        1
EL MACRO SE LLAMA ENDW           Y SU AP. AL ARCHIVO ES:       15
 TABLA DE SIMBOLOS DEF.
 EL SIMBOLO ES:ETI           Y SU VALOR ES:        5
 EL SIMBOLO ES:AUX           Y SU VALOR ES:        3
 EL SIMBOLO ES:CHANO         Y SU VALOR ES:        5
EL AP. AL ARCHIVO =      20
>@ <EOF>
>
>
>
```

```
>
>; LA SALIDA ES:
>
>
>PIP TI:=SALIDA.TMP
C       COMMENTS
        I=KIL
    2   CONTINUE
        IF (.NOT.( FLAG2 ) GO TO    3
        J=II
    4   CONTINUE
        IF (.NOT.( FLAG ) GO TO    5
                CHANO=123+CHANO
                MEMOR=JJ+MEMOR
        GO TO    4
    5   CONTINUE
        K=JJ
        L=MIS
        GO TO    2
    3   CONTINUE
        I=KIL
        END
.
>
>
>
>
>; NOTE LA ANIDACION DEL WHILE.
>
>
>
>
```

```
>
>
>
>; EJEMPLO DEL USO DE REPEAT EN FORTRAN.
>
>PIP TI:=REPEAT.TXT
%ETI=0
%MACRO REPEAT( )
%ETI=%ETI+1
%AUX=%ETI
%PUSH
%ESC    CONTINUE
%ENDM
%MACRO UNTIL(X)
%POP
        IF (.NOT.( X )) GO TO %ESC
%ENDM
%AUX=1
%ETI=1
C       COMMENTS
        PROGRAM MAIN
        I=COTA
        %REPEAT()
        J=20+COTA
        I=I+12
        %UNTIL( FLAG )
        K=COTA
        END.
>
>; SE LLAMA A MPPG
>
>@A
>PIP *.TMP;*/DE
>RUN MPPG
CAMBIO DE CARACTER DE ADVERTENCIA ?
N
USARA SU PROPIA BIBLIOTECA?
N
DEME EL NOMBRE DEL ARCHIVO FUENTE
REPEAT.TXT
 EL DIRECTORIO DE MACROS ES:
EL MACRO SE LLAMA REPEAT          Y SU AP. AL ARCHIVO ES:       1
EL MACRO SE LLAMA UNTIL           Y SU AP. AL ARCHIVO ES:       5
 TABLA DE SIMBOLOS DEF.
 EL SIMBOLO ES:ETI              Y SU VALOR ES:       2
 EL SIMBOLO ES:AUX              Y SU VALOR ES:       2
EL AP. AL ARCHIVO =       9
>@ <EOF>
>
>
```

```
>
>
>
>; LA SALIDA DE MPPG ES:
>
>
>
>PIP TI:=SALIDA.TMP
 C        COMMENTS
         PROGRAM MAIN
         I=COTA
     2   CONTINUE
         J=20+COTA
         I=I+12
         IF (.NOT.( FLAG )) GO TO     2
         K=COTA
         END
.
>
>
>
>
>
```

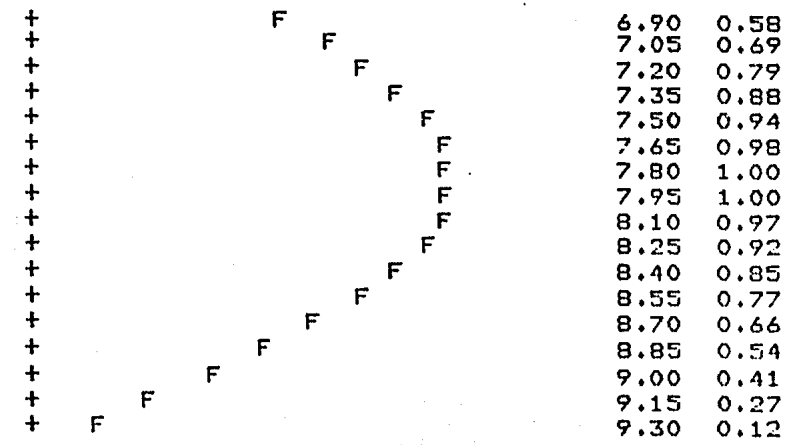
```
>
>
>; EJEMPLO FINAL DE MPPG
>;       GRAFICA.
>
>PIP TI:=FOREST.LBR
%MACRO FOREST( )
%ETI=1
%AUX=1
%MACRO WHILE( Y )
%ETI=%ETI+1
%AUX=%ETI
%PUSH
%ESC    CONTINUE
%ETI=%ETI+1
%AUX=%ETI
        IF (.NOT.( Y )) GO TO %ESC
%POP
%PUSH
%CHANO=%ETI
%ETI=%AUX
%PUSH
%ETI=%CHANO
%ENDM
%MACRO ENDW( )
%POP
        GO TO %ESC
%POP
%ESC    CONTINUE
%ENDM
%ENDM
>
>
>
>
>; EL PROGRAMA EN FORTRAN ES:
>
>PIP TI:=GRAFICA.FTN
%FOREST( )
C       PROGRAMA PRUEBA
C         GRAFICACION
C
        LOGICAL*1 VD(60),BLANC,CARAC,EJEX
        DATA BLANC,EJEX,CARAC/' ','+','F'/,NULE,NULS/5,5/,PI/3.1416/
        WRITE(NULS,10)
        JJ=01
        %WHILE( [ JJ .LT. 1000 ] )
        DO 1 II=1,60
1       VD(II)=BLANC
        AUX=JJ-II
        AUX1=AUX/100
        Y1=SIN(AUX1)
        Y2=25*Y1+28
        K=Y2
        VD(K)=CARAC
        VD(27)=EJEX
        WRITE(NULS,11)  VD, AUX1, Y1
        JJ=JJ+15
        %ENDW()
10      FORMAT(1H1,'EJE VERTICAL',60('+'))
11      FORMAT(1X,60A1,1X,F5.2,1X,F5.2)
        STOP
        END
```

```
 @A
>PIP *.TMP;*/DE
>RUN MPPG
CAMBIO DE CARACTER DE ADVERTENCIA ?
N
USARA SU PROPIA BIBLIOTECA?
Y
DEME EL NOMBRE DE SU BIBLIOTECA
FOREST.LBR
DEME EL NOMBRE DEL ARCHIVO FUENTE
GRAFICA.FTN
 EL DIRECTORIO DE MACROS ES:
EL MACRO SE LLAMA FOREST          Y SU AP. AL ARCHIVO ES:       1
EL MACRO SE LLAMA WHILE           Y SU AP. AL ARCHIVO ES:      25
EL MACRO SE LLAMA ENDW            Y SU AP. AL ARCHIVO ES:      39
 TABLA DE SIMBOLOS DEF.
 EL SIMBOLO ES:ETI              Y SU VALOR ES:       3
 EL SIMBOLO ES:AUX              Y SU VALOR ES:       3
 EL SIMBOLO ES:CHANO            Y SU VALOR ES:       3
EL AP. AL ARCHIVO =       44
>@ <EOF>
>
>
>; LS  SALIDA DE MPPG ES:
>
>PIP TI:=SALIDA.TMP
C       PROGRAMA PRUEBA
C         GRAFICACION
C
        LOGICAL*1 VD(60),BLANC,CARAC,EJEX
        DATA BLANC,EJEX,CARAC/' ','+','F'/,NULE,NULS/5*0/,PI/3.1416/
        WRITE(NULS,10)
        JJ=1
    2   CONTINUE
        IF (.NOT.( JJ .LT. 1000 )) GO TO    3
        DO 1 II=1,60
 1       VD(II)=BLANC
        AUX=JJ-II
        AUX1=AUX/100
         Y1=SIN(AUX1)
        Y2=25*Y1+28
         K=Y2
        VD(K)=CARAC
        VD(27)=EJEX
        WRITE(NULS,11)  VD, AUX1, Y1
        JJ=JJ+15
        GO TO    2
    3   CONTINUE
10      FORMAT(1H1,'EJE VERTICAL',60('+'))
11      FORMAT(1X,60A1,1X,F5.2,1X,F5.2)
        STOP
        END
>
>; ESTE PROGRAMA SE EJECUTARA
>;
>FOR GRAF=SALIDA.TMP
.MAIN.
>TKB GRAF=GRAF
>
```

```
>
>RUN GRAF




EJE VERTICAL++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
            F             +                                  -0.60 -0.56
                F         +                                  -0.45 -0.43
                   F      +                                  -0.30 -0.30
                       F  +                                  -0.15 -0.15
                          +F                                  0.00  0.00
                          +   F                               0.15  0.15
                          +       F                           0.30  0.30
                          +          F                        0.45  0.43
                          +              F                    0.60  0.56
                          +                 F                 0.75  0.68
                          +                   F               0.90  0.78
                          +                     F             1.05  0.87
                          +                       F           1.20  0.93
                          +                        F          1.35  0.98
                          +                        F          1.50  1.00
                          +                        F          1.65  1.00
                          +                        F          1.80  0.97
                          +                       F           1.95  0.93
                          +                     F             2.10  0.86
                          +                   F               2.25  0.78
                          +                F                  2.40  0.68
                          +             F                     2.55  0.56
                          +          F                        2.70  0.43
                          +       F                           2.85  0.29
                          +   F                               3.00  0.14
                          +                                   3.15 -0.01
                       F  +                                   3.30 -0.16
                   F      +                                   3.45 -0.30
               F          +                                   3.60 -0.44
            F             +                                   3.75 -0.57
         F                +                                   3.90 -0.69
       F                  +                                   4.05 -0.79
     F                    +                                   4.20 -0.87
   F                      +                                   4.35 -0.94
  F                       +                                   4.50 -0.98
  F                       +                                   4.65 -1.00
  F                       +                                   4.80 -1.00
  F                       +                                   4.95 -0.97
   F                      +                                   5.10 -0.93
     F                    +                                   5.25 -0.86
       F                  +                                   5.40 -0.77
          F               +                                   5.55 -0.67
             F            +                                   5.70 -0.55
                F         +                                   5.85 -0.42
                    F     +                                   6.00 -0.28
                       F  +                                   6.15 -0.13
                          +F                                  6.30  0.02
                          +    F                              6.45  0.17
                          +       F                           6.60  0.31
                          +           F                       6.75  0.45
```


6.90 0.58
7.05 0.69
7.20 0.79
7.35 0.88
7.50 0.94
7.65 0.98
7.80 1.00
7.95 1.00
8.10 0.97
8.25 0.92
8.40 0.85
8.55 0.77
8.70 0.66
8.85 0.54
9.00 0.41
9.15 0.27
9.30 0.12

```
TT6  --  STOP
>
>
>
>
>
```

CAP VI CONCLUSIONES.

En Mexico se desarrolla principalmente software para resolver los problemas especificos que se presentan en este momento. Sin embargo, el desarrollo de programas de servicio es muy esporadico ya que estos programas constituyen un trabajo que necesita mayor tiempo y experiencia.

La gran mayoria del software de servicio que se utiliza en el pais es de procedencia extranjera, comunmente de los Estados Unidos de Norte America.

El presente Macro Procesador de Proposito General representa un esfuerzo en la direccion del desarrollo de software de servicio de gran aplicacion. Este Macro procesador es de proposito general, es decir, el uso es independiente del lenguaje de programacion utilizado, lo cual permite gran flexibilidad y poderio.

Podemos resumir a MPPG como un preprocesador de texto de proposito general que admite textos con definiciones de Macros y produce textos con los Macros expandidos. MPPG admite ademas bibliotecas de Macros, cambio de caracteres de control, anidacion de Macro-definiciones y Macro-llamadas. Mas aun, en la anidacion de Macro-llamadas es permitida la recursion y para esto es tambien realizada la expansion condicional de texto.

MPPG aumenta su poder al permitir entrada / salida controlada, de tal forma que el usuario pueda dar a MPPG texto, donde texto puede ser desde una constante hasta una Macro-definicion.

El presente trabajo es producto de una investigacion sobre Macro procesadores ya realizados pero que carecen de algunas caracteristicas importantes. En MPPG se conjuntaron todas las caracteristicas positivas de los Macro procesadores construidos hasta ahora que fueren faciles de entender y manejar.

MPPG como todo software de servicio debera pasar por un proceso de depuracion, es decir, un proceso en el cual usuarios comunes y corrientes hagan uso de el, de tal manera que salgan a relucir los pequenos detalles que solamente pueden verse con un uso continuo del software.

BIBLIOGRAFIA

BROWN

BROWN, P. J.
THE ML/I MACRO PROCESSOR.
COMMUNICATION ACM
VOL. 10, NO 10, OCTOBER 1967.

BROWN-2

BROWN, P. J.
ML/I USER'S MANUAL
UNIVERSITY OF KENT AT CANTERBURY
4TH EDITION.

BORWN-3

BROWN, P. J.
MACRO PROCESSORS
JOHN WILEY SON, 1975.

FLETCHER-1

FLETCHER, J. G.
A PROGRAM TO SOLVE THE PENTOMINO
PROBLEM BY THE RECURSIVE USE OF
MACROS.
COMMUNICATION ACM
VOL. 8,NO 10.

GRIES-1

GRIES, DAVID.
COMPILER CONSTRUCTION FOR DIGITAL
COMPUTERS.

LEVINE-1

LEVINE, GUILLERMO.
FOREST:PROCESADOR PARA EXTENDER FORTRAN.
MEMORIAS DE LA CONFERENCIA SOBRE MICRO-
COMPUTADORAS Y MICROPROCESADORES.
FUNDACION ARTURO R., ABRIL 1980.

MOOERS-1

MOOERS, C.
TRAC, A TEXT HANDLING LANGUAGE.
PROCEDING 20TH ACM NATIONAL CONFERENCE.

NICHOLLS-1 NICHOLLS, J.
PL/I COMPILE TIME EXTENSIBILITY.
SIGPLAN NOTICES.
VOL. 4, NO 8.

STRACHEY-1 STRACHEY, C.
A GENERAL PURPOSE MACROGENERATOR.
COMPUTER JOURNAL.
VOL. 8, NO 3.

ULLMAN-1 ULLMAN, JEFFREY.
FUNDAMENTAL CONCEPS OF PROGRAMMING SYSTEMS.
ADDISON WESLEY.

WAITE WAITE, W. M.
A LANGUAGE IDEPENDENT MACRO PROCESSOR.
COMMUNICATION ACM
VOL. 10, NO 7.

WIRTH-1 WIRTH, NIKLAUS.
PASCAL USERS MANUAL AND REPORT.
SPRINGER VERLAG 1974.

WIRTH-2 WIRTH, NIKLAUS.
ALGORITHMS + DATA STRUCTURES = PROGRAMS.
PRENTICE HALL, 1976.