UNIVERSIDAD PONTIFICIA COMILLAS

ESCUELA UNIVERSITARIA DE INFORMATICA
ESPECIALIDAD DE GESTION

PROYECTO FIN DE CARRERA

COMPILADOR

DIRECTOR: **JUAN ANTONIO PEREZ CAMPANERO**

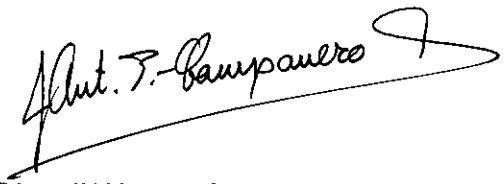
AUTOR: **VICTOR DEL BARRIO TERCEÑO**

Autorizada la entrega del proyecto del alumno

VICTOR DEL BARRIO TERCEÑO

Madrid 3 de Junio de 1.994

EL DIRECTOR DEL PROYECTO



Fdo.: JUAN ANTONIO PEREZ CAMPANERO

RESUMEN

Un compilador es un programa que traduce un lenguaje en otro. En este caso, un programa escrito en un lenguaje de alto nivel se convierte en lenguaje ensamblador. El ensamblador y el lincador no son considerados como parte de un compilador.
La estructura típica de un compilador es la siguiente:

La primera parte es el analizador léxico, que agrupa los caracteres del texto fuente en grupos con entidad propia denominados tokens, siendo un token una unidad lexica indivisible. La segunda parte es el analizador sintáctico, un grupo de rutinas que convierte una cadena de tokens en un árbol sintáctico. El árbol sintáctico representa la sentencia de una forma jerárquica, pasando de una visión general a sus partes constituyentes. En tercer lugar se encuentra el analizador semántico, que analiza la semántica de las sentencias consultando unas tablas auxiliares denominadas tablas de símbolos que recogen las características de todas las variables utilizadas en el programa. Por último, el generador de código genera el lenguaje de bajo nivel.

ABSTRACT

A compiler is a program that translates one languaje into another. In this case, the source code of a high-level computer languaje is translated into assembly languaje. The assembler and the linker are not considered to be part of the compiler. The structure of a typical compiler is this:

The first part is the lexical analizer, wich looks at the input stream as a collection of basic languaje elemets called tokens. That is, a token is a indivisible lexical unit. The second part is the parser, a group of subroutines that converts a token stream into a parse tree, and a parse tree is a structural representation of the sentence being parsed. The parse tree represents the sentence in a hierarchical fashion, moving from a general description of the setence down to the specific sentence being parsed at the leaves. And the third part, the code generator, generates the low-level computer languaje.

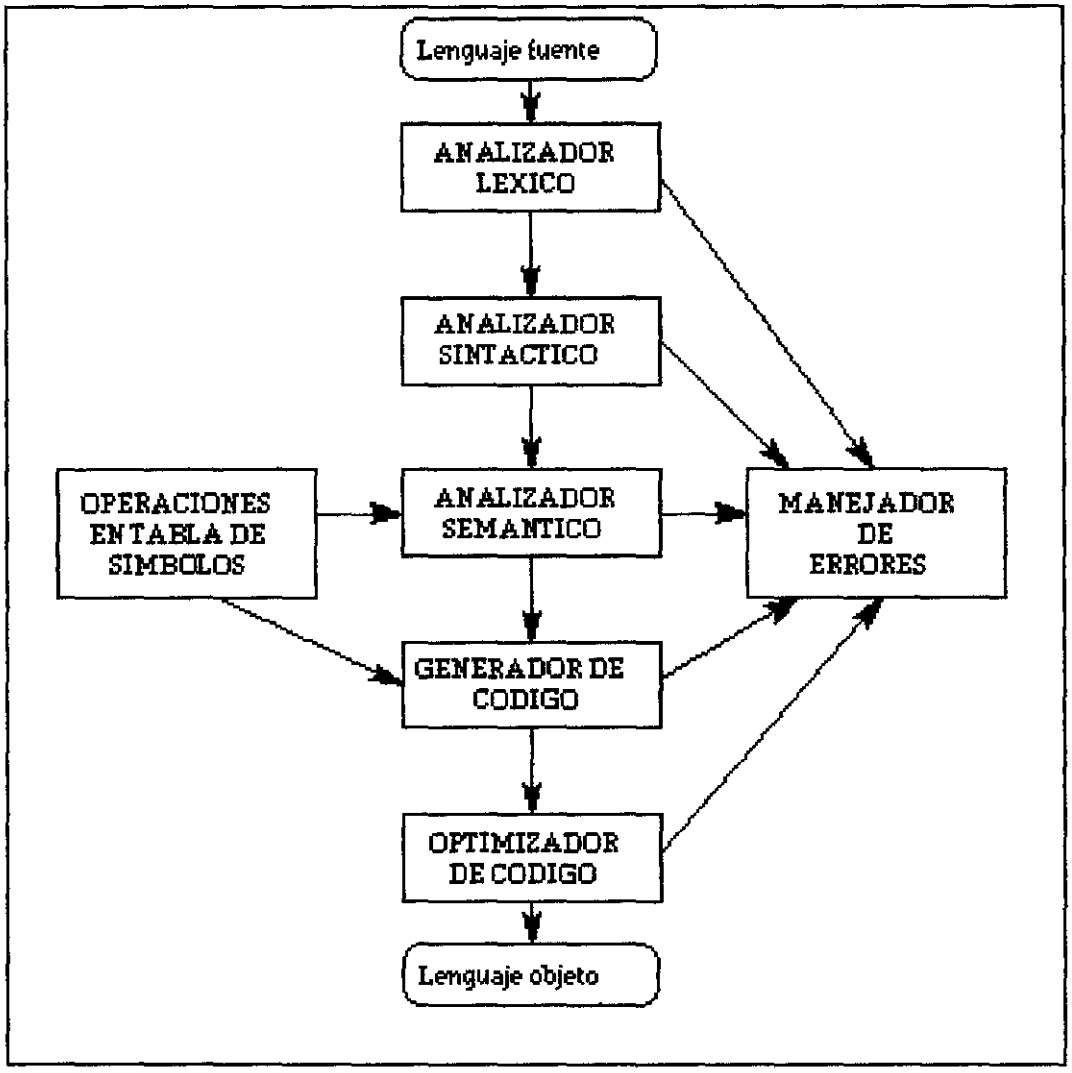
INDICE

1.- COMPILADORES

1.1.- Conceptos básicos

Desde el punto de vista informático, un lenguaje es una notación formal para describir algoritmos o funciones que serán ejecutadas por un ordenador. Actualmente, existe una gran variedad de lenguajes. Ahora bien, todos ellos tienen un punto en común. Su gramática. Con ella podremos saber si una sentencia escrita para un determinado lenguaje es valida o no. Y ello lo podemos determinar mediante un Compilador. Así pues podemos definir un compilador como el programa que convierte un lenguaje de alto nivel en otro comprensible para la maquina, aplicando una serie de reglas especificas de ese lenguaje, esto es, su gramática.

Esta tarea de transformación se puede llevar a cabo de dos formas: mediante un interprete o mediante un compilador. El funcionamiento del primero consiste en ejecutar una a una las sentencias que componen el programa. Esto cuenta con la ventaja de que permite la interacción con el programa durante la ejecución, con lo que se pueden hacer cambios en el mismo y observar los resultados inmediatamente, facilitando enormemente la depuración. Como desventaja, es muy lento. Con el compilador sin embargo la ejecución es mucho mas rápida pues después del proceso de compilación se ha creado un fichero ejecutable equivalente. En realidad, la salida de un compilador además de lo mencionado, puede ser otro lenguaje de alto nivel, un lenguaje ensamblador, que es el caso que nos ocupa y un lenguaje maquina reubicable. Hoy en día la mayoría de los compiladores suelen producir por defecto un fichero ejecutable sobre el que se puede aplicar diversas optimizaciones (disminución de tamaño del ejecutable, eliminar código redundante, incrementar la velocidad de ejecución...) e integran funciones de depuración que hacen las veces de interprete, por lo que en un mismo paquete tenemos integradas las ventajas de ambos sistemas. La estructura clásica de un compilador lleva asociadas una serie de fases encadenadas :



Además de los procesos descritos, en un compilador hay otras actividades a realizar, de las que destacamos dos: el control de las tablas de símbolos y el tratamiento de errores.

Un compilador necesita guardar y usar la información de los objetos que va encontrando en el lenguaje fuente como variables, etiquetas, declaraciones de tipos, etc. Esta información se va introduciendo en una estructura de datos interna al compilador conocida con el nombre de tabla de símbolos. El conjunto de procedimientos para el manejo de esta tabla, como introducir un nuevo símbolo, consultar la información de un símbolo, modificarla, borrarla, etc,. es lo que se denomina control de las tablas de símbolos. En cuanto al tratamiento de errores, es el conjunto de rutinas y actividades que tratan la identificación de un error, su posible tratamiento o recuperación y la emisión del mensaje correspondiente. Como puede verse en el gráfico, habrá un conjunto de errores para cada fase del compilador en cuestión.

Para tener una idea mas clara de los tipos de errores posibles asociados a cada fase, veamos los siguientes ejemplos:
En la frase,

EL perrro corre por el cammmpo

un error de tipo léxico correspondería a las palabras perrro y cammmpo
En la frase,

La comer es buena

un error de tipo sintáctico corresponde a la palabra comer , pues en el sujeto de una oración un verbo no puede acompañar a un artículo.
Por ultimo, en la frase,

La rueda come patatas

todo es correcto (léxica y sintacticamente hablando) menos su sentido.

Aparte de estos errores, cuya existencia impediría la creación de código, existen otros dos tipos que solo se detectarían en ejecución;

- Lógicos. Responden a una mala programación. Pueden dar lugar a bucles infinitos, a segmentos de código que nunca se ejecutan...
- De ejecución. Responden al entorno en el que se ejecutan. Pueden dar lugar a desbordamientos de tablas, falta de memoria para continuar...

1.2.- Análisis léxico

El Análisis léxico tiene como misión agrupar los caracteres del texto fuente en grupos con entidad propia denominados tokens (signos lingüísticos). Los tokens pueden ser palabras reservadas, nombres de variables, signos especiales, numeros. Actúa normalmente como un procedimiento que es llamado por el analizador sintáctico cuando este necesita un nuevo token. El analizador léxico devuelve, al analizador sintáctico que lo llamó, una representación del token que ha encontrado en el texto fuente. Para ello habrá saltado los espacios y comentarios que pudieran haber delante del token.

Un algoritmo muy potente que nos servirá para detectar y localizar palabras reservadas e identificadores se basa en la búsqueda dicotómica. Si la búsqueda tiene éxito, el token es una palabra reservada; si no, es un identificador. Su funcionamiento es el siguiente:

Supongamos que queremos localizar un determinado token en una lista de palabras reservadas previamente ordenada. Evidentemente la forma de buscarlo no es secuencialmente. Por el contrario será mas efectivo abrir la lista por el medio para determinar que mitad contiene el token buscado. A continuación abrimos la lista elegida por el medio y así volver a determinar que cuarta parte de la misma lo contiene. Si reiteramos este proceso podremos encontrar el token buscado rápidamente, puesto que reducimos rápidamente las posibles zonas donde se encuentra el token buscado. Veamos esto en un ejemplo practico, donde queremos encontrar la letra I:

A
B
C
D
E
F
G
H <------ 1
I <------- 4
J <------- 3
K
L <------ 2
M
N
O

1.3.- Análisis sintáctico

Se ocupa de analizar la sintaxis de las sentencias (compuestas de tokens), de acuerdo con la descripción sintáctica reflejada en la gramática. La definición de la sintaxis se realiza con una notación denominada gramática independiente del contexto. Una gramática describe de forma natural la estructura jerárquica de muchas construcciones de los lenguajes de programación. Por ejemplo, una proposición if-else en C tiene la forma

if (expresión) proposición else proposición

Esto es, la proposición es la concatenación de la palabra clave if, unos paréntesis que delimitan una expresión, una proposición, la palabra clave else y otra proposición. Esta regla de estructuración se expresa

prop -> if (expr) prop else prop

donde es posible leer la flecha como "puede tener la forma". Dicha regla se denomina producción. En una producción, los elementos léxicos como la palabra clave if y los paréntesis, se denominan componentes léxicos. Las variables expr y prop representan secuencias de componentes léxicos y se llaman no terminales.

Una gramática independiente del contexto tiene cuatro componentes:

- Un conjunto de componentes léxicos, denominados símbolos terminales.
- Un conjunto de no terminales.
- Un conjunto de producciones, en el que cada producción consta de un no terminal, llamado lado izquierdo de la producción, una fecha y una secuencia de componentes léxicos y no terminales, o ambos llamados lado derecho de la producción.
- La denominación de uno de los no terminales como símbolo inicial.

Se sigue la regla convencional de especificar las gramáticas dando una lista de sus producciones, donde las producciones del símbolo inicial se listan primero.

Para realizar este trabajo contamos con una utilidad muy potente y ampliamente difundida: el generador de analizadores sintácticos YACC ("Yet Another Compiler Compiler", otro compilador de compiladores mas). Esta utilidad va a servir para definir la sintaxis de un nuevo lenguaje. Un programa fuente en YACC tiene tres partes:

declaraciones
%%
reglas de traducción
%%
rutinas en C de apoyo

La parte de declaraciones. Hay dos secciones opcionales. En la primera se ponen declaraciones ordinarias en C, delimitadas por %{ y %}. Aquí se sitúan las declaraciones de todas las temporales usadas por las reglas de traducción o los

procedimientos de las segunda y tercera secciones. En la segunda habrá declaraciones de los componentes léxicos de la gramática.
Los componentes léxicos que se declaran en esta sección se pueden utilizar después en la segunda y terceras partes de la especificación en YACC.

La parte de las reglas de traducción. En la parte de la especificación en YACC después del primer par %% se ponen las reglas de traducción. Cada regla consta de una producción de la gramática y la acción semántica asociada. Un conjunto de producciones que se han escrito

<lado izquierdo> -> <alt1> ¦ <alt2> ¦ ... ¦ <altn>

en YAC se escribiría,

```
<lado izquierdo> :   <alt1> {acción semántica 1}
                  ¦  <alt2> {acción semántica 2}
                .....
                  ¦  <altn> {acción semántica n}
                  ;
```

En una producción en YACC, un carácter simple entrecomillado 'c' se considera como el símbolo terminal c, y las cadenas sin comillas de letras y dígitos no declarados como componentes léxicos se consideran no terminales. Los lados derechos alternativos se pueden separar con una barra vertical, y un símbolo de punto y coma sigue a cada lado izquierdo con sus alternativas y sus acciones semánticas. El primer lado izquierdo se considera como el símbolo inicial.

Una acción semántica en YACC es una secuencia de proposiciones en C. En una acción semántica, el símbolo $$ al valor del atributo asociado con el no terminal del lado izquierdo, mientras que $i, se refiere al valor asociado con el i-esimo símbolo gramatical (terminal o no terminal) del lado derecho. La acción semántica se realiza siempre que se reduzca por la producción asociada, por lo que normalmente la acción semántica calcula un valor para $$ en función de los $i. En general, {$$ = $i; } es la acción semántica por omisión.

La parte de las rutinas de apoyo en C. Se debe proporcionar un Análisis léxico de nombre yylex(). En caso necesario se pueden agregar otros procedimientos. El analizador léxico especificado produce pares formados por un componente léxico y su valor de atributo asociado. El valor de un atributo asociado a un componente

léxico se comunica al analizador sintáctico mediante la variable yylval.

1.4. El analizador semántico

Se ocupa de analizar la semántica de las sentencias, consultando la tabla de los símbolos. Básicamente su labor consistirá en asegurar la coherencia de los tipos de las variables donde estas aparezcan. Así, por ejemplo, en un bucle For-Next hará que vigilar que ambas variable sean de tipo numérico. En esta fase no existen utilidades de ayuda como YACC para la fase sintáctica, así que su implantación se hará con rutinas asociadas siempre que haya por medio una variable.

1.5. Generado de código

Esta fase del compilador es la que se encarga de generar las instrucciones en código objeto o intermedio para un nivel dado de una maquina. Los códigos a generar pueden ser:

- Lenguaje maquina ejecutable directamente
- Lenguaje maquina reubicable
- Lenguaje ensamblador
- Lenguaje de alto nivel

Las ventajas de producir uno u otro tipo de código son las siguientes:

- Producir código maquina absoluto tiene la ventaja de poder cargar y ejecutar el programa inmediatamente.
- Producir código maquina reubicable es la solución adoptada por la mayoría de los compiladores existentes, pero precisa de un enlazador y un cargador.
- Producir ensamblador tiene la ventaja de poder utilizar instrucciones simbólicas y llamadas a macros de ensamblador. Ahora bien, hay que añadir una fase mas: el ensamblado.
- Producir un lenguaje de alto nivel simplifica mucho las cosas pues hacemos que muchos problemas los solucione el compilador de ese lenguaje.

Los dos primeros puntos se pueden discutir en entornos PC bajo MS-DOS, siendo su formato COM y EXE respectivamente. Se podría decir que los EXE son una evolución de los COM, pues la razón de ser de estos era la compatibilidad con el antiguo sistema operativo CP/M. De hecho hoy se desaconseja la creación de estos

ficheros, pues aunque dan mas trabajo al sistema operativo a la hora de ejecutarse (algo imperceptible dada la velocidad del Hardware actual), permiten programas mas extensos y mejor estructurados. Ambos tienen muchas diferencias, como se puede apreciar en la tabla siguiente:

	Ficheros COM	Ficheros EXE
Tamaño	Máximo de 64 Kb	Toda la memoria del sistema
Segmentos de memoria	El mismo como segmento de código, datos y pila	Se inicializa uno para código, datos y pila
Estructura	Imagen exacta del código tal y como se va a cargar en memoria	- Cabecera con registro iniciales - Tabla de reubicación - Programa
Pasos del DOS a la hora de ejecutar un programa	- Crear PSP - Dar valores a los registros de los segmentos	- Crear PSP - Leer cabecera - Asignar valores indicados - Reservar pila para su uso

2. DESCRIPCIÓN DEL PROTOTIPO

2.1. Elementos del lenguaje. Constantes y variables

El termino constante designa un valor especifico y determinado que se define al hacer un programa y que no cambia a lo largo del mismo. Variable hace alusión a un nombre simbólico con el que se hace referencia a un dato que puede tomar valores diversos, y **siempre** han de declararse al principio del programa. Existen dos tipos: Enteras y caracteres.

Constantes enteras. Es una sucesión de dígitos precedidos o no por el signo negativo o positivo. Su rango oscilará entre +32767 y -32767. Su forma a la hora de implantarse en un programa será: **0-n**, si el numero es negativo y **n** si es positivo. Como ejemplo vamos a convertir una sencilla operación matemática en una línea de código:

-2 + 5 * (-4 + 6 - 7) -> a = 0-2 + 5 * (0-4 + 6 - 7)

Una constante que incluya espacios en blanco entre sus dígitos omitirá los mismos. Ejemplo: 2 56 equivale a 256.

Constantes de caracteres. Es un conjunto de uno o más caracteres precedidos y seguidos por el carácter especial comilla ("). Este carácter se denomina delimitador. Un ejemplo de constante de carácter es el siguiente:

"Este es un texto de 46 caracteres de longitud"

Variables enteras. Designan una zona de memoria cuyo contenido es una constante o valor entero. Su asignación se realiza colocando el nombre de la variable, el carácter especial igual (=) y el valor deseado. Son ejemplos validos:

a = 123
b = 0-1234

Si el valor excediera los rangos antes especificados (p.e. a = -56123), la variable tomaría valores erróneos. En este punto hay que distinguir dos acciones: asignar una variable desde dentro de una línea de código e introducirla on-line. En el primer caso habría que utilizar la fórmula antes descrita (**0 - n**); en el segundo eso no seria

necesario y bastaría con introducir el signo menos (-) y el valor deseado.

Variables de caracteres. Designan una zona de memoria cuyo contenido es una constante o valor carácter y su tamaño, especificado cuando se definió, es variable.

2.2.- Sentencias

Operadores Aritméticos. A la hora de realizar operaciones matemáticas se contara con cuatro operadores aritméticos:

OPERADOR	SIGNIFICADO
/	División
*	Multiplicación
-	Resta
+	Suma

Sentencias de asignación. Hay dos tipos de sentencias de asignación: Aritméticas y Carácter.

Aritmética. Sirven para ordenar al calculador la realización de cálculos aritméticos. Su forma aritmética es:

```
v = e
```

Siendo: **v** el nombre de la variable
e una expresión Aritmética

Carácter. Sirven para relacionar una cadena de caracteres con una variable.

```
v = ec
```

Siendo: **v** el nombre de la variable
ec una expresión carácter

Sentencias Variables, Programa. Un programa se dividirá en dos partes; en la primera se definirán todas las variables que se van a utilizar, mientras que la segunda se verá ocupada por el programa propiamente dicho.

```
Variables
{
        ...
        ...
        ...
}
Programa
{
        ...
        ...
        ...
}
```

Sentencias Character, integer. Dentro del apartado de variables, podremos definir los dos tipos que hemos definido previamente; variables de caracteres y variables enteras. Su formato es el siguiente:

Character *n var1, var2, var3*m, var4

Siendo n un numero entero <= 255 que especifica el tamaño de las variables (en este caso var1, var2, var4) definidas

m un numero entero <= 255 que especifica el tamaño de la variable a la que se encuentra asociado (en este caso var3)

Por ejemplo, si quisiéramos definir 3 variables de caracteres, 2 de ellas con tamaño 5 y una con tamaño 8 deberíamos ejecutar la siguiente línea de código:

Character *5, var3*8, var1, var2

Nótese que el orden de las mismas no afecta su definición. En el caso de no especificar el valor **n**, se aceptaría como 1 por defecto.

Integer var1, var2, var3

Este caso es mas sencillo que el anterior, pues basta con especificar los nombres.

Sentencia Gotoxy. A la hora de presentar tanto un texto en pantalla, como una entrada o salida de datos, existe la opción de posicionar dicha salida en cualquier parte de la pantalla. Para ello contamos con dicha sentencia cuya sintaxis es

Gotoxy(n,m)

donde n es la fila
m es la columna

Sentencia Read. Para permitir la entrada de valores por el teclado, bien sean numéricos o de tipo carácter, se utiliza la Sentencia Read, cuya sintaxis es

Read var_n

donde var_n es el nombre de una variable.

Sentencia Print. Para permitir la salida de valores por el monitor, bien sean numéricos o de tipo carácter, se utiliza la Sentencia Print, cuya sintaxis es

Print var_n

donde var_n es el nombre de una variable.
También existe otro posible uso de esta sentencia y es con constantes de caracteres

Un ejemplo de las sentencias que hemos citado hasta ahora puede ser el siguiente:

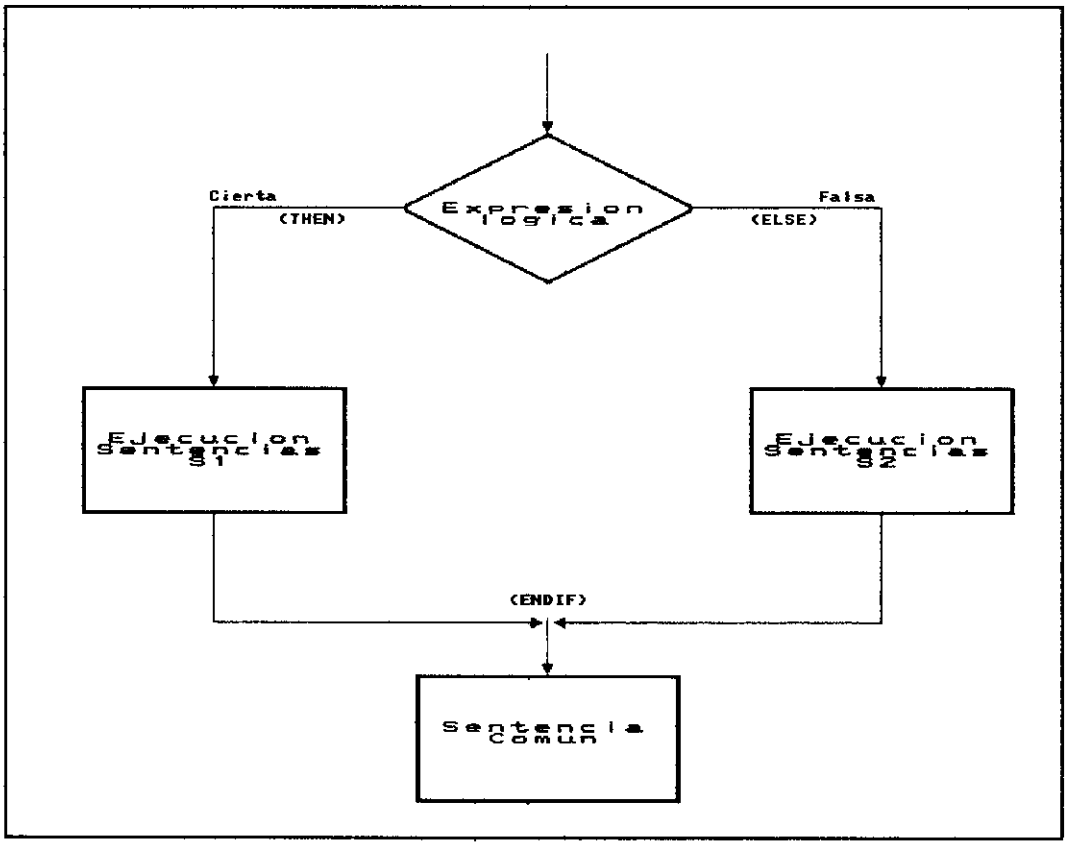
...

```
...
gotoxy(10,10)
print "Introduzca el valor A"
gotoxy(11,10)
read var1

gotoxy(12,10)
print "Introduzca el valor B"
gotoxy(13,10)
read var1

c = a + b
gotoxy(15,10)
print "El resultado de A+B es: "
gotoxy(16,10)
print C
...
...
```

Sentencia If-then-else-endif. Este conjunto de sentencias, es También reconocido como bloque IF y permite evaluar una expresión lógica, de forma que su resultado influya en el camino a tomar por el programa. Su sintaxis es la siguiente:

```
IF (e) THEN
        ..
        ..
ELSE
        ..
        ..
ENDIF
```

Y el organigrama seria el indicado en la siguiente figura:



En el bloque IF puede que no exista el conjunto de sentencias 2, pues solo se contempla el caso de que la expresión lógica sea cierta. Su sintaxis quedaría entonces reducida a:

```
IF (e) THEN
        ..
        ..
        ..
ENDIF
```

Dentro de un bloque IF pueden existir otros anidados en el. Su sintaxis seria ahora así:

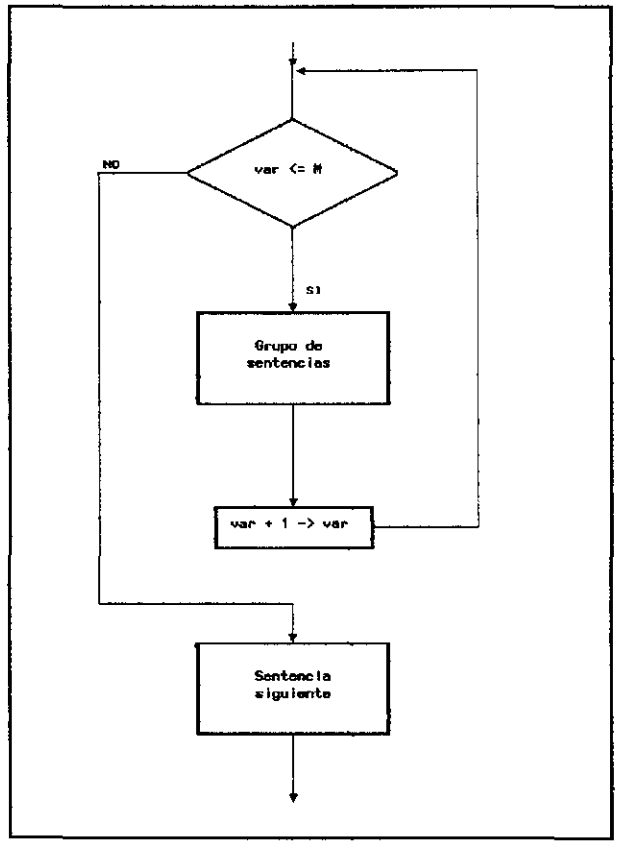
```
IF (e) THEN
        IF (e) THEN
        ELSE
        ENDIF
ELSE
ENDIF
```

Sentencia For-to-next. Para ejecutar repetidamente un determinado bloque de código, existe una sentencia especial: la sentencia FOR. Su sintaxis es:

```
FOR VAR = N TO M
    ..
    ..
NEXT
```

Donde N es un entero que indica el valor inicial
M es un entero que indica el valor final
Var es una variable definida como entera, que va tomando los valores definidos desde N hasta M.

El flujo marcado por una sentencia FOR puede indicarse mediante una el diagrama siguiente :

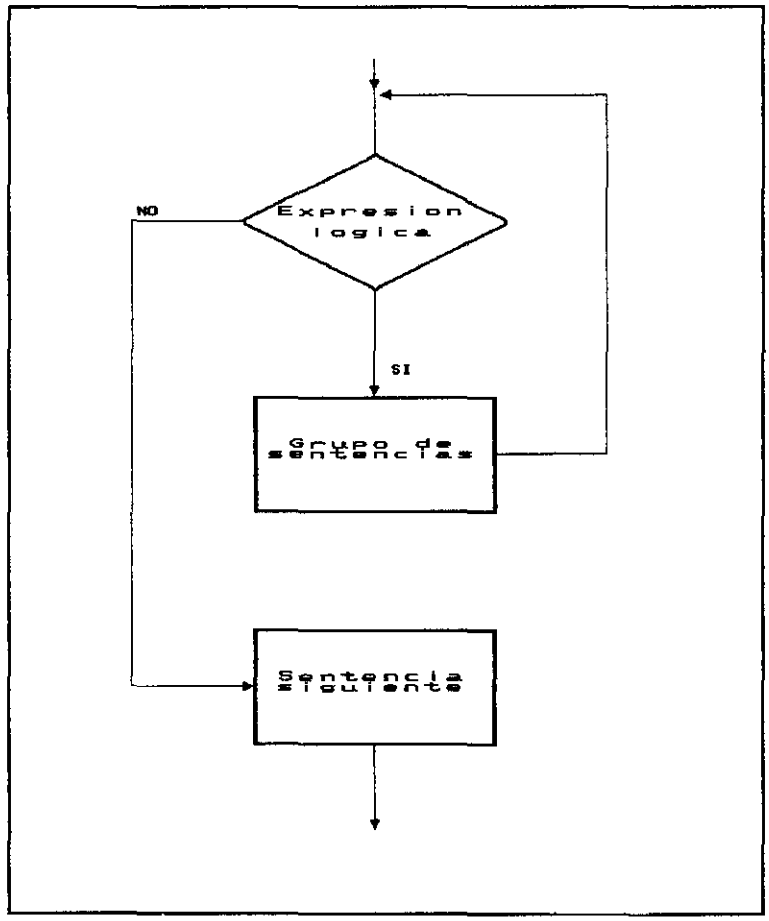


Sentencia Cls. El cometido de esta función es preparar la pantalla para una próxima salida de datos, esto es, la borra completamente.

Sentencia While-endwhile. La sentencia While, es parecida a la FOR en cuanto repite un bloque de código un numero de veces; ahora bien, mientras que esta lo hace un numero determinado de veces, aquella se rige por el resultado de una expresión lógica. Su sintaxis es la siguiente:

```
WHILE (e)
    ..
    ..
    ..
ENDWHILE
```

El flujo marcado por una sentencia WHILE puede observarse en el diagrama siguiente:



Sentencia Goto. La sentencia Goto, sirve para romper el flujo lógico de un programa. Su sintaxis es:

GOTO etiq

Donde **etiq** es un numero que representa el lugar donde va a continuar el programa.
La utilización de esta sentencia rompe el flujo lógico de un programa. Así

```
..
..
10:
..
..
GOTO 10
```

hará que el bloque de código entre la etiqueta 10 y la sentencia GOTO se repita indefinidamente.
No esta permitido su uso dentro de sentencias For o While.

2.3.- ELEMENTOS LÓGICOS

Operadores y expresiones de relación. Se denomina operador de relación un operador tal que al ser aplicado sobre dos elementos del mismo tipo, datos numéricos o carácter, produce un resultado lógico, es decir, cierto o falso. Estos operadores pueden ser los siguientes:

OPERADOR DE RELACIÓN	SIGNIFICADO
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
=	Igual que

Operadores lógicos. Como se indicó en el apartado anterior, al aplicar operadores de relación se obtienen resultados lógicos. Pues bien, para poder formar expresiones lógicas mas complejas contamos con los operadores lógicos. Estos son dos:

AND
OR

En la tabla siguiente podemos observar el resultado después de aplicar todas las combinaciones posibles:

A	B	A AND B	A OR B
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Expresiones lógicas. Una expresión lógica se emplea para expresar un determinado calculo lógico o booleano. Cuando se evalúa una expresión lógica se produce un resultado También de tipo lógico. Puede estar constituida por los siguientes elementos:

- Una expresión de relación
- Los operadores lógicos
  AND
  OR
- Cualquiera de las formas que constituyen esta relación entre paréntesis.

2.4.- ANALISIS LÉXICO

Los pasos en que se descompone la fase LEXICA son:
- Introducción por pantalla del nombre del programa a compilar. No es necesario introducir la extensión .FOR aunque si lo es que el nombre del programa cuente con ella.

- Carga en memoria de forma dinámica de todas las líneas que componen el programa. En el caso de que sea muy grande y no quepa en memoria dará un error y abortara la ejecución.
- Búsqueda y asociación de todos y cada uno de los caracteres del fichero de entrada en sus respectivos elementos de lenguaje.
- Transformación del fichero de entrada en otro intermedio de salida con extensión *.LEX que especificará los componentes léxicos del mismo. En este punto los errores que puedan darse serán principalmente por haber tecleado mal una sentencia.

Después de la ejecución de estos pasos, dará comienzo el proceso propiamente dicho del analisis léxico, que consistirá en ir devolviendo tokens a todas las llamadas de la fase sintáctica que lo requieran, para lo cual irá leyendo el archivo previamente creado (ya sin errores de tipo léxico). Esta función, llamada YYLEX(), tiene una peculiaridad como consecuencia de la necesidad de controlar el anidamiento de las sentencias repetitivas; y esta consiste en "anticiparse" al YACC, detectando ciertas palabras reservadas que indican la salida y entrada de dichas sentencias, de forma que se puedan actualizar diversas tablas que controlan los citados anidamientos.

2.5.- ANALISIS SINTÁCTICO

Como comente en su momento, esta fase ha sido realizada con la ayuda de una herramienta muy extendida: el YACC. Con ella ha sido definida la sintaxis, como se puede observar en el listado adjunto.

```
%}

%TOKEN var
%TOKEN lit
%TOKEN cons

%%
programa        :       'S' '\n' '{' cuerpo_var '}' '\n' 'T' '\n' '{' cuerpo_prog '}'
                                { comp_goto_etiq(); exit(0); }
                ;
cuerpo_var      :       cuerpo_var 'A' cuerpo2
                |       cuerpo_var 'A' '*' cons cuerpo21
                |       cuerpo_var 'B' cuerpo3
```

```
                |       cuerpo_var '\n'
                |
                ;
cuerpo2         :       cuerpo2 var '*' cons      { add_var(ult_var,1,ult_cons); }
                |       cuerpo2 var              { add_var(ult_var,1,1); }
                |       cuerpo2 ',' var          { add_var(ult_var,1,1); }
                |       cuerpo2 ',' var '*' cons  { add_var(ult_var,1,ult_cons); }
                |
                ;
cuerpo21        :       cuerpo21 var '*' cons     { add_var(ult_var,1,ult_cons); }
                |       cuerpo21 var              { add_var(ult_var,1,primer_cons); }
                |       cuerpo21 ',' var          { add_var(ult_var,1,primer_cons); }
                |       cuerpo21 ',' var '*' cons { add_var(ult_var,1,ult_cons); }
                |
                ;
cuerpo3         :       cuerpo3 var               { add_var(ult_var,2,1); }
                |       cuerpo3 ',' var           { add_var(ult_var,2,1); }
                |
                ;
cuerpo_prog     :       cuerpo_prog 'C' '(' cons ',' cons ')'
                                { comp_gotoxy(primer_cons,ult_cons);gen_gotoxy(); }
                |       cuerpo_prog 'D' var   { comp_read(ult_var); gen_read(ult_var); }
                |       cuerpo_prog 'E' var   { comp_write(ult_var); gen_write(ult_var); }
                |       cuerpo_prog 'E' lit   { gen_write_lit(); }
                |       cuerpo_prog 'F' '(' condicion ')' 'G' '\n' cuerpo_prog '\n' 'I'
                |       cuerpo_prog 'F' '(' condicion ')' 'G' '\n' cuerpo_prog '\n' 'H' '\n'
                                cuerpo_prog '\n' 'I'
                |       cuerpo_prog 'J' var '=' cons 'K' cons '\n' cuerpo_prog '\n' 'M'
                                { gen_for2(); }
                |       cuerpo_prog 'N' '(' condicion ')' '\n' cuerpo_prog '\n' 'O'
                |       cuerpo_prog 'P' cons     { comp_goto(); gen_goto(); }
                |       cuerpo_prog cons ':'     { comp_etiq(); gen_etiq(); }
                |       cuerpo_prog var '=' expr
                                { polaca_comun(); gen_asig(); limpiar_polaca(); }
                |       cuerpo_prog var '=' lit  { gen_asig2(); }
                |       cuerpo_prog 'L'          { gen_cls(); }
                |       cuerpo_prog '\n'         { sw_primer_cons = 0; error_o(); }
                |
                ;
expr            :       expr var  { swv += 1; compv(); comp_expr(ult_var); polaca_exp1(); }
                |       expr cons  { swc += 1; compc(); polaca_exp2(); }
                |       expr oper2 { swo += 1; compo(); polaca_exp3(); }
                |       expr '(' expr ')'
                |
                ;
```

```
condicion       :       condicion var oper var
                        { sw_sema_cond = 0; comp_cond(ant_var,ult_var); polaca(); }
                |       condicion '(' condicion ')'
                |       condicion oper3  { comp_cond2(); sw_sema_cond = 1; polaca1();}
                |
                ;
oper            :       '<'  { tok_oper = '1'; }
                |       '>'  { tok_oper = '2'; }
                |       'a'  { tok_oper = '3'; }
                |       'b'  { tok_oper = '4'; }
                |       '='  { tok_oper = '5'; }
                ;
oper2           :       '+'  { tok_oper = '+'; }
                |       '-'  { tok_oper = '-'; }
                |       '*'  { tok_oper = '*'; }
                |       '/'  { tok_oper = '/'; }
                ;
oper3           :       'Q'  { tok_oper = '6'; }
                |       'R'  { tok_oper = '7'; }
                ;
%%
```

Su mecánica de funcionamiento es la siguiente: Tras llamar a la función que activa realmente el proceso -YYPARSE()-, esta llama a otra -YYLEX()- que es la que lee el fichero *.LEX creado y va devolviendo uno a uno todos las palabras reservadas, variables, constantes y literales de los que constaba el programa fuente. Notese que las palabras reservadas han sido asociados a una letra en mayúscula según la siguiente tabla:

LETRA	TOKEN ASOCIADO
A	Character
B	Integer
C	Gotoxy
D	Read
E	Print
F	If
G	Then
H	Else
I	Endif
J	For

K	to
L	Cls
M	Next
N	While
O	Endwhile
P	Goto
Q	And
R	or
S	Variables
T	Programa

Para explicar el comportamiento del YACC, tomemos como ejemplo la línea de la definición de la sintaxis que controla la estructura básica del programa:

```
programa : 'S' '\n' '{' cuerpo_var '}' '\n' 'T' '\n' '{' cuerpo_prog '}' {comp_goto_etiq();exit(0);}
```

Podemos interpretarlo de la siguiente forma:

Un programa va a estar compuesto de una palabra reservada denominada 'VARIABLES', un salto de línea, un carácter corchete '{', un CUERPO_VAR donde van a estar definidas todas las variables, otro carácter corchete '}', un salto de línea, otro carácter corchete '{', la palabra reservada 'PROGRAMA', un CUERPO_PROG que contendrá el programa propiamente dicho, un salto de línea y por ultimo un corchete '}'.

Notese que tanto CUERPO_VAR como CUERPO_PROG se vuelven a definir en líneas sucesivas. Esto indica que son símbolos NO TERMINALES pues requieren una explicación mediante una regla o producción posterior. Por otra parte, nos encontramos con los símbolos TERMINALES 'S', 'T' y los corchetes. Estos tienen entidad propia y se describen por si mismos, es decir, sin ninguna producción.

En ultimo lugar, al final de la línea, podemos observar una serie de sentencias en C que se ejecutaran cuando la producción asociada se realice. Esto es muy importante y de hecho constituye la fase semántica y la de generación de código.

2.5.1.- Notación Polaca

Supongamos que queremos evaluar la siguiente expresión aritmética:

5 + 2 * (5 - 1) - 6 / 2

El primer paso consistiría en deshacer los paréntesis

5 + 2 * 4 - 6 / 2

El siguiente en operar las multiplicaciones y las divisiones

5 + 8 - 3

Y por ultimo las sumas y las restas. Pues bien, una labor tan sencilla como desarrollar una expresión aritmética, adquiere una nueva dificultad a la hora de ser tratada por el ordenador a niveles de programación cercanos a la maquina (ensamblador) a causa de que en estos niveles, no existen paréntesis y los operadores son bivalentes. La expresión aritmética mostrada antes se encontraba en modo INFIJO, esto es, con los operadores en medio de los operandos. Ahora bien, este modo es incomprensible para un ordenador y es por ello por lo que se utiliza la Notación Polaca. Su propiedad fundamental es que el orden en que se van a realizar las operaciones esta completamente determinado por las posiciones de los operadores y los operandos en la expresión; así nunca se necesitan paréntesis al establecer expresiones en Notación polaca. Por ejemplo:

+AB -EF *GR /ED

Hay otro modo, llamado Notación Polaca Inversa. Tendría el siguiente aspecto:

AB+ EF- GR* ED/

y es al que he convertido las expresiones infijas de los programas a compilar. Básicamente el algoritmo que realiza la conversión ejecuta dos pasos; en el primero se convierte la expresión infija a postfija y en el segundo se fracciona esa expresión en sus correspondientes instrucciones en ensamblador. Una estructura de datos vital para llevar a cabo este algoritmo es la Pila de datos.

Una Pila de datos es una lista de elementos en la cual un elemento puede ser insertado o eliminado por un extremo, llamado la cima de la pila. Esto significa que los elementos se sacan de la pila en orden inverso al que se insertaron en él.

Consideremos el siguiente ejemplo:

A + (B * C - (D / E ↑ F) * G) * H

Los pasos a dar para encontrar su equivalente postfija serían los siguientes:

Símbolo examinado	**PILA**	**Expresión Postfija**
A	(	A
+	(+	A
(	(+(	A
B	(+(	AB
*	(+(*	AB
C	(+(*	ABC
-	(+(-	ABC*
(	(+(-(	ABC*
D	(+(-(	ABC*D
/	(+(-(/	ABC*D
E	(+(-(/	ABC*DE
↑	(+(-(/↑	ABC*DE
F	(+(-(/↑	ABC*DEF
)	(+(-	ABC*DEF↑/
*	(+(-*	ABC*DEF↑/
G	(+(-*	ABC*DEF↑/G
)	(+	ABC*DEF↑/G*-
*	(+*	ABC*DEF↑/G*-
H	(+*	ABC*DEF↑/G*-H
)		ABC*DEF↑/G*-H*+

Tras el ultimo paso la pila quedaría vacía y la expresión resultante sería:

ABC*DEF↑/G*-H*+

2.5.2.- Tabla de símbolos

Es en este punto donde se añaden las variables que requerirá el programa. Para ello contamos con una tabla dinámica que reflejara cada variable con sus determinadas características. Su estructura es la siguiente:

Nombre	Tipo	Longitud	Puntero a siguiente elemento de la tabla

2.6.- ANALISIS SEMANTICO

Esta fase se va a centrar casi exclusivamente en la verificación de que los tipos de variables utilizados y sus usos, sean los correctos. Por ejemplo que en un bucle for se utilice una variable numérica y no carácter. Para ello me he servido, como indique en el apartado anterior, de la línea de código asociada a cada producción en YACC. De esta forma, el analisis sintáctico y semantico (y como veremos mas adelante, la generación de código) se realizan no en bloque aislados y dando una salida que sirviera de entrada para el siguiente paso como ocurriera en el analisis léxico, sino prácticamente al unísono pues en el momento en que una producción se ha cumplido, su línea de código asociada se ejecuta y comprueba que los datos utilizados son semanticamente correctos. Mención especial merecen las sentencias con bucles, pues estas pueden anidarse unas dentro de otras, hecho que dificulta su control y obliga a mantener tablas que controlen dicho anidamiento.
En la siguiente tabla se especifica todas las comprobaciones semánticas de cada sentencia:

Rutina	**Función que realiza**
COMP_VAR	Comprueba que todas las variables utilizadas hayan sido definidas previamente.
COMP_FOR	Comprueba que se utilicen variables numéricas y que los rangos del bucle no excedan de un determinado margen
COMP_GOTO	Comprueba que no exista un GOTO dentro de un bucle While o For
COMP_ETIQ	Comprueba que no existan etiquetas duplicadas y no se encuentren dentro de un bucle For o While
COMP_COND	Comprueba que las variables de una condición tengan tipo numérico

COMP_READ y COMP_WRITE	Llaman a comp_var
COMP_GOTOXY	Comprueba que las coordenadas especificadas estén dentro de unos limites
COMP_GOTO_ETIQ	Comprueba que la etiqueta a la que hace referencia un GOTO exista
COMP_V, COMP_C y COMP_O	Comprueban la buena construcción de una expresión

2.7.- GENERACIÓN DE CÓDIGO

En esta última fase, se genera el código del programa introducido. Para ello he utilizado ensamblador de la familia INTEL 80x86 y servicios del DOS siempre que he podido. Esto ultimo ha facilitado mucho la labor, pues por ejemplo para limpiar la pantalla basta con llamar a un determinado servicio de la interrupción 21.

El proceso de compilar, por tanto, requiere del proceso de ensamblado (TASM.EXE) y lincado (TLINK.EXE) adicional de un programa fuente en ensamblador (salida del compilador) con el mismo nombre que su equivalente, pero con extensión *.ASM. Automáticamente el compilador llama a estos programas después de generar este fichero. En consecuencia, es conveniente que el subdirectorio que contiene a esos programas, se encuentre en el PATH del sistema.

A continuación detallo la traducción de cada sentencia en su correspondiente conjunto de mnemonicos.

SENTENCIA	**EQUIVALENTE EN ENSAMBLADOR**

CLS	mov ax,0700h
	mov bh,07h
	mov cx,0000h
	mov dx,2479h
	int 10h

GOTOXY	mov ah,02h
	mov bh,00h
	mov dx,050ah
	int 10h

PRINT "literal"	mov ah,040h
	mov bx,0001h
	mov cx,21d
	mov dx,offset lit_1
	int 21h

READ variable	call	inic_r
	mov	ah,03Fh
	mov	bx,0000h
	mov	cx,size n_car
	add	cx,0002h
	mov	dx,offset n_car
	int	21h
	call	read_n
	mov	word ptr ax,ds:[n_car_]
	mov	word ptr ds:[variable],ax
PRINT variable	mov	ax,ds:[variable]
	call	write_n
ASIGNACION	push	10
variable = 10 + (5 - 3) * 7	push	5
	push	3
	pop	bx
	pop	ax
	sub	ax,bx
	push	ax
	push	7
	pop	bx
	pop	ax
	mul	bx
	push	ax
	pop	bx
	pop	ax
	add	ax,bx
	push	ax
	pop	ax
	mov	ds:[variable],ax

SENTENCIA IF
 	
if condicion then
  print "Then"
else
  print "Else"
endif
 	
CONDICION
(entero1 > entero3)
	mov ax,ds:[entero1]
	mov bx,ds:[entero3]
	cmp ax,bx
	ja c_1
	push 0
	jmp c_1_
	c_1: push 1
	c_1_:
	pop ax
	
	cmp ax,0001h
	je cond_1a
	
	jmp cond_1
	
	cond_1a:
	
	mov ah,040h
	mov bx,0001h
	mov cx,4d
	mov dx,offset lit_2
	int 21h
	
	jmp cond_1_
	cond_1:
	
	mov ah,040h
	mov bx,0001h
	mov cx,4d
	mov dx,offset lit_3
	int 21h
	cond_1_:

SENTENCIA FOR		mov	ds:[entero3],01h
for entero3 = 1 to 10		mov	cx,01h
entero1=entero1+1	f_j_1:	cmp	cx,0ah
next		ja	faf_1
		inc	cx
		push	cx
		call	f_1__
		inc	ds:[entero3]
		pop	cx
		jmp	f_j_1
	faf_1:	jmp	f_f_1
	f_1__	proc	near
		push	ds:[entero1]
		push	1
		pop	bx
		pop	ax
		add	ax,bx
		push	ax
		pop	ax
		mov	ds:[entero1],ax
		ret	
	f_1__	endp	
	f_f_1:		

PROCEDIMIENTO DE LECTURA	read_n proc near
	mov si,offset n_car
	add si,0005h
	mov ds:[n_car_],0
	mov cx,0001h
	aqui: xor ax,ax
	mov byte ptr al,ds:[si]
	cmp al,02Dh
	je aqui3
	cmp al,030h
	jb aqui2
	cmp al,039h
	ja aqui2
	sub ax,0030h
	mul word ptr cx
	add ds:[n_car_],ax
	xchg ax,cx
	cmp si,0000h
	je fuera2
	mov cx,0010d
	mul word ptr cx
	xchg ax,cx
	aqui2: cmp si,0000h
	je fuera2
	dec si
	jmp aqui
	aqui3: mov ax,0000h
	sub ax,ds:[n_car_]
	mov ds:[n_car_],ax
	fuera2:
	ret
	read_n endp

PROCEDIMIENTO DE ESCRTITURA	write_n proc near
	cmp ax,0000h
	jge wri1
	not ax
	add ax,01h
	push ax
	mov ah,02h
	mov dl,0C4h
	int 21h
	pop ax
	wri1: mov cx,10000d
	fuera0: push cx
	mov dx,0000h
	cmp ax,cx
	jb fuera
	dentro: inc dx
	sub ax,cx
	cmp ax,cx
	jb fuera
	jmp dentro
	fuera: pop cx
	push ax
	xchg ax,cx
	cmp ax,0001
	je fuera1
	mov bx,10d
	push dx
	mov dx,0000h
	div word ptr bx
	pop dx
	xchg ax,cx
	mov ah,02h
	add dx,030h
	int 21h
	pop ax
	jmp fuera0
	fuera1:
	pop ax
	mov ah,02h
	add dx,030h
	int 21h
	ret
	write_n endp

SENTENCIA WHILE
while condicion
  entero1 = entero1 + 1
endwhile
 	
CONDICION
entero1 < entero3 and
entero2 = entero3 or
entero2 = entero4
 	
(notese la complejidad de la condicion y su descomposicion en partes para su desarrollo)
	c_w_1:
	  mov ax,ds:[entero1]
	  mov bx,ds:[entero3]
	  cmp ax,bx
	  jb c_1
	  push 0
	  jmp c_1_
	c_1: push 1
	c_1_: mov ax,ds:[entero2]
	  mov bx,ds:[entero4]
	  cmp ax,bx
	  je c_2
	  push 0
	  jmp c_2_
	c_2: push 1
	c_2_: mov ax,ds:[entero2]
	  mov bx,ds:[entero4]
	  cmp ax,bx
	  je c_3
	  push 0
	  jmp c_3_
	c_3: push 1
	c_3_: pop ax
	  pop bx
	  or ax,bx
	  pop bx
	  and ax,bx
	
	  cmp ax,0001h
	  je cond_1a
	
	  jmp cond_1
	cond_1a:
	  push ds:[entero1]
	  push 1
	  pop bx
	  pop ax
	  add ax,bx
	  push ax
	  pop ax
	  mov ds:[entero1],ax
	
	  jmp c_w_1
	cond_1:

VARIABLES
{
..
..
}
PROGRAMA
{
..
..
..
}
	pila segment stack
	db 64 dup ('Pila')
	pila ends
	
	data segment
	n_car db 7 dup (" ")
	n_car_ dw 0
	lit_1 db "Introduzca un número "
	lit_2 db "then"
	lit_3 db "else"
	entero1 dw 0
	entero3 dw 0
	data ends
	
	codigo segment
	assume cs:codigo,ds:data,ss:pila
	nombre proc far
	mov ax,data
	mov ds,ax
	..
	..
	..
	mov ax,4C00h
	int 21h
	nombre endp
	codigo ends
	end nombre

2.7.- VALIDACION DEL COMPILADOR

Una vez desarrollado el compilador para el nuevo lenguaje, el siguiente paso consistirá en probarlo exhaustivamente a fin de detectar cualquier condición de error que pueda acontecer. Para ello se deberán generar programas de prueba cuyos resultados se conocen de antemano, y compararlo con la salida real del compilador. En el caso que nos ocupa, se podrán desarrollar prácticamente todas las pruebas necesarias para comprobarlo, sin embargo en un caso real donde los recursos económicos para esta fase son limitados, habrá que desechar todas aquellas circunstancias en las cuales haya errores, pero cuya probabilidad de aparecer en explotación tienda a cero, con el fin de no malgastar tiempo y dinero.

Donde he puesto mas énfasis a la hora de probar el funcionamiento, ha sido en tres puntos principalmente:

- Evaluación de expresiones
- Evaluación de condiciones
- Funcionamiento de sentencias repetitivas anidadas

- Evaluación de expresiones

Como ya explique en su momento, para evaluar las expresiones y traducirlas a un formato comprensible por el ensamblador, utilicé notación polaca postfija. Respecto a los operadores y al tamaño máximo de los operandos, he utilizado el formato interno del microprocesador, es decir, operandos de 2 bytes (65536) y los cuatro operadores básicos (+,-,*,/). Para una versión posterior usaré el coprocesador, pues aparte de proporcionar mucha mas precisión y cantidad de instrucciones, su uso resulta muy parecido al sistema actual, pues consta de los siguientes elementos :

Una pila de ocho elementos, cíclica
Siete tipos de datos (entero, entero corto, entero largo, decimal empaquetado, real corto, real largo, real temporal)
Entorno

El riesgo mayor de producirse un error se encontraba en respetar la prioridad de operandos y paréntesis. Después de varias pruebas con variaciones del programa que listo a continuación conseguí depurar las correspondientes rutinas en ensamblador. Con ello también descubrí, que el formato que debía adoptarse en una asignación de un numero negativo era de **0 - n**, siendo n el numero en cuestión.

```
variables
{
integer i1,i2,i3,i4,i5,i6
}
programa
{
cls
gotoxy(8,10)
read i1
i2 = 0-2700
gotoxy(10,2)
```

```
print i2
i3 = (1+(1-3)+5)*i2
gotoxy(10,10)
print i3
i4 = 1+(1*2+37/2-3)+5
gotoxy(10,18)
print i4
i5 = 1+(1*(2+37)/2-3)+5
gotoxy(10,26)
print i5
i6 = (((((((i1)))))))
gotoxy(10,34)
print i6
for i6 = 1 to 5
  i1 = i1 - 1
next
gotoxy(10,42)
print i1
i1 = i1 + 5
gotoxy(10,50)
print i1
}
```

- Evaluacion de condiciones

Tambien aquí utilicé la notacion polaca. Y gracias al programa que a continuacion listo, descubrí que, dado que decidí adjudicar tanto a AND como a OR la misma prioridad y debia utilizar parentesis para establecer la misma, dicha prioridad no se cumplia. Fue suficiente con cambiar una instruccion de extraccion de la pila por otra de introduccion, para que funcionara.

```
variables
{
integer ent1, ent2, ent3, ent4, ent5
}
programa
{
cls
gotoxy(5,10)
print "Introduzca número 1 "
read ent1
gotoxy(6,10)
print "Introduzca número 2 "
read ent2
```

```
ent3 = 7
ent4 = 7
ent5 = 7
while (ent1 = ent2 or (ent3 = ent4 and ent5 = ent4) or ent5 = ent4)
  print "w"
endwhile
while (ent1 < ent2 or ent3 = ent4 and ent5 = ent4)
  ent1 = ent1 + 1
  gotoxy(9,10)
  print "Para salir ent5 = 0"
  read ent5
endwhile
print ent1
}
```

- Evaluacion de anidamientos

El objetivo de esta prueba consistia en verificar que no hubiera conflictos entre bucles internos y externos de una misma sentencia e incluso de varias distintas. He aqui el programa base sobre el cual realicé las pruebas.

```
variables
{
integer ent1, ent2, ent3, ent4, ent5, n
}
programa
{
cls
for n = 1 to 10
  gotoxy(5,10)
  print "Número 1 "
  read ent1
  ent2 = 10
  while (ent1 < ent2)
    for ent3 = 1 to 5
        ent1 = ent1 + 1
    next
    if (ent1 < ent2) then
        print "Then"
    else
        print "Else"
        for ent4 = 1 to 4
          ent1 = ent1 - 1
        next
    endif
```

```
  endwhile
next
}
```

Por último, señalar que también hice pruebas para averiguar el tamaño máximo del fichero de entrada y que este depende de la versión del compilador con la que compile mi proyecto , siendo las mas modernas las que además de compilar, permiten llamadas a los programas externos TASM y TLINK.

3.- CONCLUSIONES

Cuando cursaba primero de carrera y mientras esperaba paciente que después de mandar una compilación de algún programa FORTRAN, mi terminal devolviera el mensaje de compilación correcta, me entretenía pensando como estaría trabajando en ese preciso instante el ordenador con mi programa y como traduciría cada sentencia a otras de mas bajo nivel. Probablemente si el Host hubiera tardado menos no me encontraría ahora escribiendo estas líneas. Pues bien, ya desde esos instantes, comenzó a gestarse en mi mente la idea de como trabajaba un compilador. Ya había trabajado con ensamblador haciendo pequeñas rutinas y veía que el sistema operativo MS-DOS, ofrecía por su diseño, muchas facilidades para implementar ciertas funciones básicas de todo lenguaje tales como borrar pantalla, posicionar el cursor en un lugar determinado, imprimir una cadena de caracteres ... incluso tenía un buen nivel para manejar ficheros. Todo esto me animó y me ayudó para tomar la decisión definitiva de hacerlo. Sin embargo, esa sencillez fue un espejismo; al poco tiempo de meterme de lleno en la tarea, y tomar conciencia de la complejidad del tema y a pesar de que contaba con la inestimable ayuda del YACC, comenzaron a aparecer dificultades que hicieron que mis estimaciones iniciales de tiempo se vieran desbordadas. Todo esta dificultad se puede observar en los listados que adjunto en los apéndices, donde es fácil apreciar el trabajo que conlleva la mayor parte del código del programa y lo preciso que debe ser el manejo de ciertos switchs (por ejemplo los que controlan el nivel de anidamiento de las sentencias repetitivas) para que no se produzcan situaciones inesperadas.

Respecto a la experiencia adquirida, el desarrollo de este tema me ha proporcionado un mejor conocimiento del sistema operativo MS-DOS, del ensamblador y sobre todo me ha servido para llenar un gran interés hacia un tema fascinante que no fue tratado en los cursos de la universidad.

Hoy, con el ya acabado, no me arrepiento en absoluto de tomar aquella decisión y puedo decir que lo realmente difícil fue el principio, pero una vez que comencé fue cuestión de echarle imaginación y ganas. Quizás mi hobbie futuro sea ir implementando mas y mas funciones hasta hacer un compilador medianamente serio.

4.- BIBLIOGRAFÍA

COMPILADORES E INTERPRETES

G. Sanchez Dueñas, J.A. Valverde Andreu

Ediciones Diaz de Santos, S.A.

COMPILADORES. PRINCIPIOS, TECNICAS Y HERRAMIENTAS

Alfred V. Aho, Ravi Sethi, Jeffrey D.Ullman

Addison-Wesley iberoamericana

COMPILADORES. TEORIA Y CONSTRUCION

Sanchis Llorca, Galan Pascual

Paraninfo

PROGRAMACION ENSAMBLADOR EN ENTORNO MS-DOS

Miguel Angel Rodriguez-Roselló

Anaya

SISTEMA OPERATIVO DOS 4

J. de Yraolagoitia

Paraninfo

ESTRUCTURA DE DATOS

Seymour Lipschutz

Mc Graw Hill

5.- APENDICES

5.1.- Compilación de programa ejemplo

Para detallar lo explicado hasta ahora adjunto tres listados de un programa ejemplo en los que se puede observar la evolución del programa ejemplo introducido, desde el formato .FOR hasta el .ASM, pasando por el .LEX.

Proyecto.FOR

```
variables
{
character *20 caract1
integer entero1, entero2, entero3, entero4
}
programa
{
5:
cls
gotoxy(5,10)
print "Introduzca un número "
read entero1
entero3 = 10
entero4 = 0
if ((entero1 > entero3) or (entero1 < entero4)) then
  goto 5
else
  for entero2 = 1 to 10
    entero1 = entero1 + 1
  next
  gotoxy(7,10)
  print "El numero introducido + 10 es "
  print entero1
  gotoxy(9,10)
  print "Escribamos * "
  print entero1
  print " veces "
  entero3 = 1
  while (entero1 >= entero3)
    print " *"
    entero1 = entero1 - 1
  endwhile
endif
gotoxy(19,10)
print "Introduzca un texto (Máx. 20) "
read caract1
gotoxy(20,10)
print "El texto introducido fue "
print caract1
}
```

Proyecto.LEX

```
S
{
A * cons_20 var_caract1
B var_entero1 , var_entero2 , var_entero3 , var_entero4
}
T
{
cons_5 :
L
C ( cons_5 , cons_10 )
E lit "Introduzca un número "

D var_entero1
var_entero3 = cons_10
var_entero4 = cons_0
F ( ( var_entero1 > var_entero3 ) R ( var_entero1 < var_entero4 ) ) G
P cons_5
H
J var_entero2 = cons_1 K cons_10
var_entero1 = var_entero1 + cons_1
M
C ( cons_7 , cons_10 )
E lit "El numero introducido + 10 es "

E var_entero1
C ( cons_9 , cons_10 )
E lit "Escribamos * "

E var_entero1
E lit " veces "

var_entero3 = cons_1
N ( var_entero1 >= var_entero3 )
E lit " *"

var_entero1 = var_entero1 - cons_1
O
I
C ( cons_19 , cons_10 )
E lit "Introduzca un texto (Máx. 20) "

D var_caract1
C ( cons_20 , cons_10 )
E lit "El texto introducido fue "

E var_caract1
}
```

Proyecto.ASM

```
pila    segment stack
        db 64 dup ('Pila')
pila    ends

data    segment
        n_car  db      7       dup (" ")
        n_car_         dw      0
        lit_1   db     "Introduzca un número "
        lit_2   db     "El numero introducido + 10 es "
        lit_3   db     "Escribamos * "
        lit_4   db     " veces "
        lit_5   db     " *"
        lit_6   db     "Introduzca un texto (Máx. 20) "
        lit_7   db     "El texto introducido fue "
        caract1        db      20      dup (" ")
        entero1        dw      0
        entero2        dw      0
        entero3        dw      0
        entero4        dw      0
data    ends

codigo          segment
        assume cs:codigo,ds:data,ss:pila
proyecto        proc far
        mov     ax,data
        mov     ds,ax
etiq_5:
        mov     ax,0700h
        mov     bh,07h
        mov     cx,0000h
        mov     dx,2479h
        int     10h

        mov     ah,02h
        mov     bh,00h
        mov     dx,050ah
        int     10h

        mov     ah,040h
        mov     bx,0001h
        mov     cx,21d
        mov     dx,offset lit_1
        int     21h

        call    inic_r
        mov     ah,03Fh
        mov     bx,0000h
        mov     cx,size n_car
        add     cx,0002h
```

```
        mov     dx,offset n_car
        int     21h

        call    read_n
        mov     word ptr ax,ds:[n_car_]
        mov     word ptr ds:[entero1],ax

        push    10
        pop     ax
        mov     ds:[entero3],ax

        push    0
        pop     ax
        mov     ds:[entero4],ax

        mov     ax,ds:[entero1]
        mov     bx,ds:[entero3]
        cmp     ax,bx
        ja      c_1
        push    0
        jmp     c_1_
c_1:    push 1
c_1_:
        mov     ax,ds:[entero1]
        mov     bx,ds:[entero4]
        cmp     ax,bx
        jb      c_2
        push    0
        jmp     c_2_
c_2:    push 1
c_2_:
        pop     ax
        pop     bx
        or      ax,bx

        cmp     ax,0001h
        je      cond_1a

        jmp     cond_1
cond_1a:
        jmp     etiq_5

        jmp     cond_1_
cond_1:
        mov     ds:[entero2],01h
        mov     cx,01h
f_j_1:  cmp     cx,0ah
        ja      faf_1
        inc     cx
        push    cx
        call    f_1__
```

```
        inc     ds:[entero2]
        pop     cx
        jmp      f_j_1
faf_1:  jmp     f_f_1

f_1__   proc    near
        push    ds:[entero1]
        push    1
        pop     bx
        pop     ax
        add     ax,bx
        push    ax
        pop     ax
        mov     ds:[entero1],ax

        ret
f_1__   endp
f_f_1:
        mov     ah,02h
        mov     bh,00h
        mov     dx,070ah
        int     10h

        mov     ah,040h
        mov     bx,0001h
        mov     cx,30d
        mov     dx,offset lit_2
        int     21h

        mov     ax,ds:[entero1]
        call    write_n

        mov     ah,02h
        mov     bh,00h
        mov     dx,090ah
        int     10h

        mov     ah,040h
        mov     bx,0001h
        mov     cx,13d
        mov     dx,offset lit_3
        int     21h

        mov     ax,ds:[entero1]
        call    write_n

        mov     ah,040h
        mov     bx,0001h
        mov     cx,7d
        mov     dx,offset lit_4
        int     21h
```

```
        push   1
        pop    ax
        mov    ds:[entero3],ax
c_w_1:
        mov    ax,ds:[entero1]
        mov    bx,ds:[entero3]
        cmp    ax,bx
        jae    c_3
        push   0
        jmp    c_3_
c_3:    push 1
c_3_:
        pop    ax

        cmp    ax,0001h
        je     cond_2a

        jmp    cond_2
cond_2a:
        mov    ah,040h
        mov    bx,0001h
        mov    cx,2d
        mov    dx,offset lit_5
        int    21h

        push   ds:[entero1]
        push   1
        pop    bx
        pop    ax
        sub    ax,bx
        push   ax
        pop    ax
        mov    ds:[entero1],ax

        jmp    c_w_1
cond_2:
cond_1_:
        mov    ah,02h
        mov    bh,00h
        mov    dx,130ah
        int    10h

        mov    ah,040h
        mov    bx,0001h
        mov    cx,30d
        mov    dx,offset lit_6
        int    21h

        mov    ah,03Fh
        mov    bx,0000h
        mov    cx,size caract1
```

```
        add     cx,0002h
        mov     dx,offset caract1
        int     21h

        mov     ah,02h
        mov     bh,00h
        mov     dx,140ah
        int     10h
        mov     ah,040h
        mov     bx,0001h
        mov     cx,25d
        mov     dx,offset lit_7
        int     21h
        mov     ah,040h
        mov     bx,0001h
        mov     cx,size caract1
        mov     dx,offset caract1
        int     21h

        mov     ax,4C00h
        int     21h
proyecto        endp

write_n         proc near
        cmp     ax,0000h
        jge     wri1
        not     ax
        add     ax,01h
        push    ax
        mov     ah,02h
        mov     dl,0C4h
        int     21h
        pop     ax
wri1:   mov     cx,10000d
fuera0:         push    cx
        mov     dx,0000h
        cmp     ax,cx
        jb      fuera
dentro:         inc     dx
        sub     ax,cx
        cmp     ax,cx
        jb      fuera
        jmp     dentro
fuera:  pop     cx
        push    ax
        xchg    ax,cx
        cmp     ax,0001
        je      fuera1
        mov     bx,10d
        push    dx
        mov     dx,0000h
```

```
        div     word ptr bx
        pop     dx
        xchg    ax,cx
        mov     ah,02h
        add     dx,030h
        int     21h
        pop     ax
        jmp     fuera0
fuera1:
        pop     ax
        mov     ah,02h
        add     dx,030h
        int     21h
        ret
write_n         endp

read_n          proc near
        mov     si,offset n_car
        add     si,0005h
        mov     ds:[n_car_],0
        mov     cx,0001h
aqui:   xor     ax,ax
        mov     byte ptr al,ds:[si]
        cmp     al,02Dh
        je      aqui3
        cmp     al,030h
        jb      aqui2
        cmp     al,039h
        ja      aqui2
        sub     ax,0030h
        mul     word ptr cx
        add     ds:[n_car_],ax
        xchg    ax,cx
        cmp     si,0000h
        je      fuera2
        mov     cx,0010d
        mul     word ptr cx
        xchg    ax,cx
aqui2:  cmp     si,0000h
        je      fuera2
        dec     si
        jmp     aqui
aqui3:  mov     ax,0000h
        sub     ax,ds:[n_car_]
        mov     ds:[n_car_],ax
fuera2:
        ret
read_n          endp
inic_r  proc near
        mov     si,offset n_car
        mov     word ptr ds:[si],"  "
```

```
        mov     word ptr ds:[si+2],"  "
        mov     word ptr ds:[si+4],"  "
        mov     byte ptr ds:[si+6]," "
        ret
inic_r  endp
codigo ends
end     proyecto
```

5.2.- Listado del proyecto

A continuacion, el listado del proyecto, con formato de entrada para el YACC. Despues de pasarselo daria lugar a otro programa en C, pero con todas las reglas sintacticas traducidas y con rutinas que realizarian dicho analisis.

```
%{
#include<stdio.h>
#include<conio.h>
#include<io.h>
#include<string.h>
#include<process.h>
#include<stdlib.h>

int busc_tok(char *);     /* definicion de prototipos de funciones */
void add_var(char *, int, int);
void comp_gotoxy(int,int);
void comp_read(char*);
void comp_write(char*);
void comp_for(int,int);
void comp_goto(void);
void comp_etiq(void);
void comp_var(char*);
void comp_goto_etiq(void);
void comp_cond2(void);
void compv(void);
void compc(void);
void compo(void);
void error_o(void);
void gen_cls(void);
void gen_for(void);
void gen_for2(void);
void gen_gotoxy(void);
void gen_write_lit(void);
void gen_write_n(void);
void gen_read_n(void);
void gen_inic_r(void);
void gen_if(void);
void gen_asig(void);
```

```
void gen_asig2(void);
void gen_if0(void);
void gen_etiq(void);
void gen_goto(void);
void polaca(void);
void polaca1(void);
void polaca_comun(void);
void polaca_exp1(void);
void polaca_exp2(void);
void polaca_exp3(void);
void limpiar_polaca(void);
void yyerror(char*);
yyparse(void);
yylex(void);

/*      tokens soportados por el lenguaje */
char *tokens[20] = {"character","integer","gotoxy","read","print","if",
                    "then","else","endif","for","to","cls","next","while",
                    "endwhile","goto","and","or","variables","programa"
                    };

unsigned int num_li;    /* numero de linea */
int sw_lleno;           /* controla que el fichero tenga o no contenido */
int sw_palres;          /* permite distinguir variable,token o numero y caracter especial */
int sw_salir;           /* indica salir cuando caracter de comentario */
int sw_ends = 0;        /* indica el segmento que se esta cerrando (datos, codigo) */
int nn;                 /* numero de token */
int n2;                 /* bucles sin trascendencia */
int n3;                 /* bucles sin trascendencia */
int sw_1;               /* controla mayor o igual y menor o igual */

int sw_yy = 0;          /* utilizado para leer fichero *.LEX */
int sw_primer_cons = 0;         /* permite cargar una unica vez la variable primer_cons */
int sw_tabla_var = 0;   /* para crear las estructuras dinamicas de la tabla de variables */
int sw_tabla_lit = 0;   /* para crear las estructuras dinamicas de la tabla de literales */
int sw_L = 0;           /* analizar la semantica de un for (L = TO) */
int sw_for = 0;         /* indica en que momento nos encontramos dentro de un for */
int sw_while = 0;       /* indica en que momento nos encontramos dentro de un while */
int indice1 = 0;        /* recorre la variable tok_res para aislar elementos del lenguaje */
int ind_goto = 0;       /* para moverse en la tabla de goto */
int ind_etiq = 0;       /* para moverse en la tabla de etiq */
```

```
int primer_cons = 0;    /*  primera constante a la hora de definir variables de caracteres */
int ult_cons = 0;       /*  ultima constante a la hora de definir variables de caracteres */
int num_linea = 0;      /* numero de linea */
int tipo = 0;           /* tipo de variable encontrada en tabla de variables */
int longi= 0;           /* longitud de variable encontrada en tabla de variables */
int tab_goto[128];      /* tabla de goto */
int tab_etiq[128];      /* tabla de etiquetas */
int cont_lit;           /* contador de literales para formar variables */
int cont_for = 0;       /* contador de bucles FOR para formar variables */
int cont_if = 0;        /* contador de IF para formar variables */
int cont_cond = 0;      /* contador de condiciones de IF para formar variables */
int cont_while = 0;     /* contador de condiciones de WHILE para formar variables */
int tab_for[16];        /* tabla de anidamientos FOR */
int tab_while[16];      /* tabla de anidamientos WHILE */
int tab_w_if[16];       /* tabla de anidamientos WHILE + IF*/
int tabla_h[16];        /* tabla de anidamientos para ELSE */
int sw_if2 = 1;         /* para saber si estamos dentro de la condicion de un if */
int ind_p;              /* para recorrer TABLA_OPE */
int s_p_tok = 0;        /* indica en recorrido de li_tok si primera vez tok_oper =6 o 7 */
int sw_if = 0;          /* indica en que momento nos encontramos dentro de un if */
int sw_O = 0;           /* indica si ha pasado por un WHILE */
int sw_sema_cond = 0;       /* indica si AND y OR estan juntos */
int swv = 0;            /* controla si dos variables juntas en expresion */
int swc = 0;            /* controla si dos constantes juntas en expresion */
int swo = 0;            /* controla si dos operadores juntos en expresion */
int num_ent = 0;            /* numero de veces que ha entrado en : compv,c,o */

char exec[32] = "";             /* ejecutar ensamblado de programa */
char exec2[32] = "";            /* ejecutar ensamblado de programa */
char li_polaca[255] = "";       /* contiene la linea polaca */
char tabla_ope[127] = "";       /* contiene las operaciones de polaca */
char tok_oper;                  /* contiene el operando de polaca */
char ult_var[16] = "";          /* contiene la ultima variable introducida */
char ant_var[16] = "";          /* contiene la penultima variable introducida */
char bak_var[16] = "";          /* contiene la variable de asignacion */
char li_tok2[200] = "";         /* registro de fichero *.LEX */
char tok_res[255] = "";         /* contiene variable o constante o literal*/
char tok_res2[255] = "";        /* contiene variable o constante o literal (ya es conocido) */
char tok_res_limp[255] = "";    /* para limpiar variable */
char *pun = NULL;               /* puntero a linea leida desde *.LEX */
char *pun_c1 = NULL;            /* punteros para extraer elementos */
```

```
char *pun_c2 = NULL;            /*     de la linea leida */
char *p_tok = NULL;             /* recorre li_polaca token a token */

char fich_for[13];              /* nombre de fichero de entrada */
char fich_for_asm[13];          /* nombre de fichero de salida en ensamblador */
char fich_tmp[256];             /* linea leida de fichero de entrada */
char li_tok2[200];      /* resultado de traducir linea de entrada en elementos del lenguaje */
char li_asm[200];               /* linea a escribir en fichero ensamblador */
char *pun1,*pun2,*pun3,*n;      /* punteros a linea de entrada. Extraen elementos del lenguaje */
char pal[16];                   /* variable, token o numero extraido de la linea */
char pal2[16];                  /* variable o numero extraido de la linea */
char nom_prog[8];               /* nombre del programa para el segmento de codigo en ensamblador
*/
char char_tam[3];               /* tamaño de la variable a definir en fichero ensamblador */
char uso[16];                   /* uso general utlizada para cambiar de tipo numerico a caracter */
char uso2[16];                  /* idem */
char uso3[16];                  /* idem */
char uso4[16];                  /* idem */

/*      estructura que soporta una linea de codigo del fichero de entrada *.FOR */
struct p_linea {
        unsigned int num_li;
        char linea_for[80];
        struct p_linea *l1;
        };
typedef struct p_linea LINEA;
LINEA *li1,*li2,*li3;       /* punteros a estructura */

/*      estructura que contiene la tabla de variables */
struct t_var {
        char nombre[16];
        int tipo;
        int longi;
        struct t_var *v1;
        };
typedef struct t_var VAR;
VAR *p_v1,*p_v2,*p_v3;

/*      estructura que contiene la tabla de literales */
struct t_lit {
        char c1[256];
```

```
        struct t_lit *l1;
        };
typedef struct t_lit VAR_L;
VAR_L *p_l1,*p_l2,*p_l3;
FILE *file1;    /* puntero a fichero *.FOR */
FILE *file2;    /* puntero a fichero *.LEX */
FILE *file3;    /* puntero a fichero *.LEX */
FILE *file4;    /* puntero a fichero *.ASM */

%}

%TOKEN var
%TOKEN lit
%TOKEN cons

%%
programa    :   'S' '\n' '{' cuerpo_var '}' '\n' 'T' '\n' '{' cuerpo_prog '}'
                    { comp_goto_etiq(); exit(0); }
            ;
cuerpo_var  :   cuerpo_var 'A' cuerpo2
            |   cuerpo_var 'A' '*' cons cuerpo21
            |   cuerpo_var 'B' cuerpo3
            |   cuerpo_var '\n'
            |
            ;
cuerpo2     :   cuerpo2 var '*' cons        { add_var(ult_var,1,ult_cons); }
            |   cuerpo2 var                 { add_var(ult_var,1,1); }
            |   cuerpo2 ',' var             { add_var(ult_var,1,1); }
            |   cuerpo2 ',' var '*' cons    { add_var(ult_var,1,ult_cons); }
            |
            ;
cuerpo21    :   cuerpo21 var '*' cons       { add_var(ult_var,1,ult_cons); }
            |   cuerpo21 var                { add_var(ult_var,1,primer_cons); }
            |   cuerpo21 ',' var            { add_var(ult_var,1,primer_cons); }
            |   cuerpo21 ',' var '*' cons   { add_var(ult_var,1,ult_cons); }
            |
            ;
cuerpo3     :   cuerpo3 var                 { add_var(ult_var,2,1); }
            |   cuerpo3 ',' var             { add_var(ult_var,2,1); }
            |
            ;
```

```
cuerpo_prog   :   cuerpo_prog 'C' '(' cons ',' cons ')'
                      { comp_gotoxy(primer_cons,ult_cons);gen_gotoxy(); }
              |   cuerpo_prog 'D' var              { comp_read(ult_var); gen_read(ult_var); }
              |   cuerpo_prog 'E' var              { comp_write(ult_var); gen_write(ult_var); }
              |   cuerpo_prog 'E' lit              { gen_write_lit(); }
              |   cuerpo_prog 'F' '(' condicion ')' 'G' '\n' cuerpo_prog '\n' 'I'
              |   cuerpo_prog 'F' '(' condicion ')' 'G' '\n' cuerpo_prog '\n' 'H' '\n' cuerpo_prog
'\n' 'I'
              |   cuerpo_prog 'J' var '=' cons 'K' cons '\n' cuerpo_prog '\n' 'M' {gen_for2(); }
              |   cuerpo_prog 'N' '(' condicion ')' '\n' cuerpo_prog '\n' 'O'
              |   cuerpo_prog 'P' cons              { comp_goto(); gen_goto(); }
              |   cuerpo_prog cons ':'              { comp_etiq(); gen_etiq(); }
              |   cuerpo_prog var '=' expr      {polaca_comun();gen_asig();limpiar_polaca();}
              |   cuerpo_prog var '=' lit           { gen_asig2(); }
              |   cuerpo_prog 'L'                   { gen_cls(); }
              |   cuerpo_prog '\n'                  { sw_primer_cons = 0; error_o(); }
              |
              ;
expr          :   expr var        { swv+=1;compv(); comp_expr(ult_var); polaca_exp1();}
              |   expr cons                    { swc += 1; compc(); polaca_exp2(); }
              |   expr oper2                   { swo += 1; compo(); polaca_exp3(); }
              |   expr '(' expr ')'            { }
              |
              ;
condicion     :   condicion var oper var
                      {sw_sema_cond=0; comp_cond(ant_var,ult_var); polaca();}
              |   condicion '(' condicion ')'   { }
              |   condicion oper3         {comp_cond2(); sw_sema_cond = 1; polaca1();}
              |
              ;
oper          :   '<'  { tok_oper = '1'; }
              |   '>'  { tok_oper = '2'; }
              |   'a'  { tok_oper = '3'; }
              |   'b'  { tok_oper = '4'; }
              |   '='  { tok_oper = '5'; }
              ;
oper2         :   '+'  { tok_oper = '+'; }
              |   '-'  { tok_oper = '-'; }
              |   '*'  { tok_oper = '*'; }
              |   '/'  { tok_oper = '/'; }
              ;
```

```
oper3           :       'Q'  { tok_oper = '6'; }
                |       'R'  { tok_oper = '7'; }
                ;
%%
void main(int argc, char *argv[])
{
    clrscr(); strcpy(exec,"tasm ");strcpy(exec2,"tlink ");
    sw_lleno = 0;sw_tabla_lit = 0;
    if (argc == 1)
    {
        gotoxy(1,22);
        puts("Fichero fuente : ");scanf("%s",fich_for); strcat(exec,fich_for);
        strcat(exec2,fich_for);
    }
    else
    {
        strcpy(fich_for,argv[1]); strcat(exec,argv[1]); strcat(exec2,argv[1]);
    }
    if ((strcmp(strstr(fich_for,".for"),".for") !=0) && (strcmp(strstr(fich_for,".for"),".FOR") != 0))
    {
        if (strlen(fich_for) >= 9)
        {
          fich_for[8]='\0';
        }
        strcat(fich_for,".for");
    }
    if ((file1 = fopen(fich_for,"rt")) == NULL)
    {
        clrscr();gotoxy(1,22);printf("Imposible abrir fichero %s",fich_for);exit(1);
    }
    else
    {
        clrscr();
        num_li = 0; gotoxy(20,16);printf("Cargando fichero fuente.");
        gotoxy(20,18);printf("Numero de linea :");
        do
        {
          num_li += 1;strcpy(fich_tmp,"\n");
          fgets(fich_tmp,256,file1);
          if (strcmp(fich_tmp,"\n") == 0)
          {
```

```
            num_li -= 1;
            continue;
        }
        sw_lleno = 1;
        if (strlen(fich_tmp) > 80)
        {
            clrscr();printf("Linea %d contiene mas de 80 caracteres ",num_li);
            printf("%d",strlen(fich_tmp));exit(1);
        }
        li2 = (LINEA*)malloc(sizeof(LINEA));li2->l1 = NULL;
        if (li2 == NULL)
        {
            clrscr();puts("Imposible cargar fichero fuente. No hay memoria");exit(1);
        }

        if (li1 == NULL)
        {
            li1 = li2;li3 = li1;
        }
        else
        {
            li1->l1 = li2;li1 = li2;
        }
        li1->num_li = num_li;strcpy(li1->linea_for,fich_tmp);
        gotoxy(38,18);printf("%i",num_li);
    } while(!feof(file1));fclose(file1);
}
if (sw_lleno == 0)
{
    clrscr();puts("Fichero de entrada vacio !!");exit(1);
}

li1 = li3;
fich_for[strlen(fich_for)-3] = '\0';
strcat(fich_for,"lex");
if ((file2 = fopen(fich_for,"wt")) == NULL)
{
    clrscr();gotoxy(1,22);printf("Imposible abrir fichero %s",fich_for);exit(1);
}

clrscr();
```

```
gotoxy(20,16);
printf("Realizando análisis léxico."); gotoxy(20,18);printf("Numero de linea :");

do
{
    strcpy(li_tok2,"");
    pun1 = li1->linea_for;pun2 = pun1;
    do
    {
      for(; strncmp(pun2," ",1) == 0; pun2++);
      pun3 = pun2;sw_palres = 0;
      while (((*pun3>=65) && (*pun3<=90)) || ((*pun3>=97) && (*pun3<=122)) ||
            ((*pun3>=48) && (*pun3<=57)))
      {
         pun3++;sw_palres = 1;
      }

      if (sw_palres == 0)
      {
         pun3++;
      }
      if ((pun3 - pun2) > 16)
      {
         clrscr();puts("Nombre de variable demasiado grande.");
         printf(" %s  %i","Num. de linea : ",li1->num_li);exit(1);
      }
      else
      {
         if ((*pun2 == '>') || (*pun2 == '<'))
         {
                pun2++;sw_1 = 0;
                if (*pun2 == '=')
                {
                      pun3++;sw_1 = 1;

                }
                pun2--;
         }
         strcpy(pal,"                ");
         strncpy(pal,pun2,pun3-pun2);
      }
```

```
if (sw_palres == 1)
{
   if (busc_tok(pal) == 1)
   {
         *n = nn + 65;n++;*n = '\0';n--;
         strcat(li_tok2,n);strcat(li_tok2," ");
   }
   else
   {
      strcpy(pal2,"");nn = 0;
      for(; (((pal[nn] >= 48) && (pal[nn] <= 57)) || ((pal[nn] >= 97) &&
           (pal[nn] <= 122))) ; nn++);
      strncpy(pal2,pal,nn);
      pal2 [nn] = '\x0';
      if ((*pun2 >= 48) && (*pun2 <= 57))
      {
            strcat(li_tok2,"cons_");
            strcat(li_tok2,pal2);
            strcat(li_tok2," ");
      }
      else
      {
            strcat(li_tok2,"var_");
            strcat(li_tok2,pal2);
            strcat(li_tok2," ");
      }
   }
}
else
{
   sw_salir = 0;
   switch(*pal)
   {
      case '(':strcat(li_tok2,"( ");break;
      case ')':strcat(li_tok2,") ");break;
      case '[':strcat(li_tok2,"[ ");break;
      case ']':strcat(li_tok2,"] ");break;
      case '{':strcat(li_tok2,"{ ");break;
      case '}':strcat(li_tok2,"} ");break;
      case '#':sw_salir = 1;break;
      case '>':if(sw_1 == 1)
```

```
        {
          strcat(li_tok2,">= ");break;
        }
        else
        {
          strcat(li_tok2,"> ");break;
        }
case '<':if(sw_1 == 1)
        {
          strcat(li_tok2,"<= ");break;
        }
        else
        {
          strcat(li_tok2,"< ");break;
        }
case '"': for(; (*pun3 != 34) && (strcmp(pun3,"\n") != 0); pun3++);
        if(strncmp(pun3,"\n",1) == 0)
        {
          clrscr();puts("Literal no balanceado.");
          printf(" %s  %i","Num. de linea : ",li1->num_li);exit(1);
        }
        else
        {
          strcat(li_tok2,"lit ");strcat(li_tok2,pun2);
          if (sw_tabla_lit == 0)
          {
              sw_tabla_lit = 1;
              p_l3 = (VAR_L *) malloc(sizeof(VAR_L));p_l1 = p_l3;
          }
          else
          {
              p_l2 = (VAR_L *) malloc(sizeof(VAR_L));
              p_l3 -> l1 = p_l2; p_l3 = p_l2;
          }
          p_l3 -> l1 = NULL;
          strcpy(p_l3 -> c1,pun2);
        }
        if (strcmp(pun3,"\n") != 0)
        {
              pun3++;
        }
```

```
                    break;
                case '=':strcat(li_tok2,"= ");break;
                case '!':strcat(li_tok2,"! ");break;
                case '+':strcat(li_tok2,"+ ");break;
                case '-':strcat(li_tok2,"- ");break;
                case '*':strcat(li_tok2,"* ");break;
                case '/':strcat(li_tok2,"/ ");break;
                case ',':strcat(li_tok2,", ");break;
                case '.':strcat(li_tok2,". ");break;
                case ':':strcat(li_tok2,": ");break;
                case '\t':break;
                default:clrscr();puts("Caracter no reconocido");
                        printf(" %s  %i","Num. de linea : ",li1->num_li);exit(1);
            }
        }
        if (sw_salir == 1) { break; }
        pun2 = pun3;
    } while (strcmp(pun3,"\n") != 0);
    if (sw_salir == 0)
    {
            strcat(li_tok2,"\n");fputs(li_tok2,file2);
    }

    li1 = li1->l1;
    gotoxy(38,18);printf("%i",li1->num_li);
} while(li1 != NULL);
fcloseall();
strcpy(fich_for_asm,fich_for);
fich_for_asm[strlen(fich_for_asm)-3] = '\0';
strcpy(nom_prog,fich_for_asm);
nom_prog[strlen(nom_prog)-1] = '\0';
strcat(fich_for_asm,"asm");
if ((file4 = fopen(fich_for_asm,"wt")) == NULL)
{
    clrscr();gotoxy(1,22);printf("Imposible abrir fichero %s",fich_for_asm);exit(1);
}
clrscr();
gotoxy(20,16);printf("Generando codigo."); gotoxy(20,18);printf("Numero de linea :");
strcpy(li_asm,"pila\tsegment stack\n");fputs(li_asm,file4);
strcpy(li_asm,"\tdb 64 dup ('Pila')\n");fputs(li_asm,file4);
strcpy(li_asm,"pila\tends\n");fputs(li_asm,file4);
```

```
    strcpy(li_asm,"\ndata\tsegment\n");fputs(li_asm,file4);
    strcpy(li_asm,"\tn_car\tdb\t7\tdup (\" \")\n");fputs(li_asm,file4);
    strcpy(li_asm,"\tn_car_\tdw\t0\n");fputs(li_asm,file4);
    cont_lit = 1;p_l2 = p_l1;
    while (p_l2 -> c1 != NULL)
    {
        itoa(cont_lit,uso,10);
        strcpy(li_asm,"\tlit_");strcat(li_asm,uso);
        strcat(li_asm,"\tdb\t");strcat(li_asm,p_l2 -> c1);fputs(li_asm,file4);
        p_l2 = p_l2 -> l1;cont_lit += 1;
    }
    cont_lit = 0; yyparse();
}

busc_tok(char *arg1)
{
        for (nn = 0; nn < 20; nn++)
        {
                if (strncmp(tokens[nn],strlwr(arg1),strlen(tokens[nn])) == 0)
                {
                        return(1);
                }
        }
        return(0);
}

void yyerror(char *arg1)
{
        clrscr();printf("%s %d","Error sintactico.  Numero de linea : ",num_linea); exit(1);
}
```

Rutinas semanticas de comprobacion de tipos

```
/*      funcion que comprueba la existencia de una variable en la tabla de variables */
void comp_var(char *arg1)
{
        int sw_found = 0; p_v2 = p_v3;
        while (p_v2 != NULL)
        {
                if (strcmp(strlwr(p_v2->nombre),strlwr(arg1)) == 0)
                {
```

```
                    sw_found = 1; tipo = p_v2->tipo; longi = p_v2->longi;
                }
                p_v2 = p_v2->v1;
        }
        if (sw_found == 0)
        {
                clrscr();
                printf("%s %d","Error semantico. Variable no encontrada. ",num_linea);
                exit(1);
        }
}

void comp_for(int arg1, int arg2)
{
        comp_var(ult_var);
        if (tipo != 2)
        {
                clrscr();
                printf("%s %d","Error semantico. Tipo de variable incorrecto. ",num_linea);
                exit(1);
        }
        if ((arg1 > arg2) || (arg1 < 1) || (arg2 > 32000))
        {
                clrscr();
                printf("%s %d","Error semantico. Enteros fuera de margen. ",num_linea);
                exit(1);
        }
}

void comp_goto(void)
{
        int indi1 = 0,sw_temp = 0;
        for (;indi1 < ind_goto;indi1++)
        {
                if (tab_goto[indi1] == ult_cons)
                {
                        sw_temp = 1;
                }
        }
        if (sw_temp == 0)
        {
```

```
                tab_goto[ind_goto] = ult_cons;ind_goto += 1;
        }
        if ((sw_for != 0) || (sw_while != 0))
        {
                clrscr();
                printf("%s %d","Error semantico. Goto dentro de for o while. ",num_linea);
                exit(1);
        }
}

void comp_etiq(void)
{
        int indi1 = 0,sw_temp = 0;
        for (;indi1 < ind_etiq;indi1++)
        {
                if (tab_etiq[indi1] == ult_cons)
                {
                        sw_temp = 1;
                }
        }
        if (sw_temp == 0)
        {
                tab_etiq[ind_etiq] = ult_cons;ind_etiq += 1;
        }
        else
        {
                clrscr();
                printf("%s %d","Error semantico. Etiqueta duplicada. ",num_linea);
                exit(1);
        }

        if ((sw_for != 0) || (sw_while != 0) || (sw_if != 0))
        {
                clrscr();
                printf("%s %d","Error semantico. Etiqueta dentro de for , while o if.
                        ",num_linea);
                exit(1);
        }
}

void comp_cond2(void)
```

```
{
    if (sw_sema_cond == 1)
    {
        clrscr();printf("%s %d","Error semantico en condicion. ",num_linea);exit(1);
    }
}

void comp_cond(char *arg1, char *arg2)
{
        int sw_found = 0,sw_found2 = 0,tip1 = 0,tip2 = 0;
        p_v2 = p_v3;
        while (p_v2 != NULL)
        {
                if (strcmp(strlwr(p_v2->nombre),strlwr(arg1)) == 0)
                {
                        sw_found = 1;tip1 = p_v2->tipo;
                }
                p_v2 = p_v2->v1;
        }
        p_v2 = p_v3;
        while (p_v2 != NULL)
        {
                if (strcmp(strlwr(p_v2->nombre),strlwr(arg2)) == 0)
                {
                        sw_found2 = 1;tip2 = p_v2->tipo;
                }
                p_v2 = p_v2->v1;
        }
        if ((sw_found == 0) || (sw_found2 == 0))
        {
                clrscr();
                printf("%s %d","Error semantico. Variable no encontrada. ",num_linea);
                exit(1);
        }
        else
        {
                if (tip1 != tip2)
                {
                        clrscr();
                        printf("%s %d","Error semantico. Distintos tipos en condicion.
                                ",num_linea);
```

```
                exit(1);
        }
        if (tip1 == 1)
        {
                clrscr();
                printf("%s %d","Error semantico. Tipo Caracter no admitido en
                        condicion. ",num_linea);
                exit(1);
        }
    }
}

/*      funcion que comprueba que read hace alusion a una variable existente */
void comp_read(char *arg1)
{
    comp_var(arg1);
}

/*      funcion que comprueba que write hace alusion a una variable existente */
void comp_write(char *arg1)
{
    comp_var(arg1);
}

void comp_gotoxy(int arg1, int arg2)
{
    if ((arg1 < 1) || (arg1 > 25))
    {
        clrscr();printf("%s %d","Error semantico en gotoxy. ",num_linea);exit(1);
    }
    if ((arg2 < 1) || (arg2 > 80))
    {
        clrscr();printf("%s %d","Error semantico en gotoxy. ",num_linea);exit(1);
    }
}

void add_var(char *arg1,int arg2,int arg3)
{
    if (sw_tabla_var == 0)
    {
        sw_tabla_var = 1;
```

```
            p_v3 = (VAR *) malloc(sizeof(VAR));p_v1 = p_v3;
            p_v1->v1 = NULL;
        }
        else
        {
            p_v2 = p_v3;
            while (p_v2 != NULL)
            {
                if (strcmp(strlwr(p_v2->nombre),strlwr(arg1)) == 0)
                {
                    clrscr();
                    printf("%s %d","Variables duplicadas. ",num_linea);
                    exit(1);
                }
                p_v2 = p_v2->v1;
            }
            p_v2 = (VAR *) malloc(sizeof(VAR));p_v1->v1 = p_v2;p_v1 = p_v2;
            p_v1->v1 = NULL;
        }
        strcpy(p_v1->nombre,arg1);
        p_v1->tipo = arg2;
        p_v1->longi = arg3;
        if (arg2 == 1)
        {
          itoa(arg3,char_tam,10);
          strcpy(li_asm,"\t"); strcat(li_asm,arg1); strcat(li_asm,"\tdb\t");
          strcat(li_asm,char_tam); strcat(li_asm,"\tdup (\" \")\n");
          fputs(li_asm,file4);
        }
        else
        {
          strcpy(li_asm,"\t"); strcat(li_asm,arg1); strcat(li_asm,"\tdw\t"); strcat(li_asm,"0\n");
          fputs(li_asm,file4);
        }
}

void comp_expr(char *arg1)
{
        comp_var(arg1);
        if (tipo != 2)
        {
```

```
                clrscr();
                printf("%s %d","Error semantico. Tipo de variable incorrecto. ",num_linea);
                exit(1);
        }
}

void comp_goto_etiq(void)
{
        int indi1 = 0,indi2,sw_found;
        clrscr();puts("Compilacion finalizada.\n\n");
        for(; indi1 < ind_goto; indi1++)
        {
                sw_found = 0;indi2 = 0;
                for (; indi2 < ind_etiq; indi2++)
                {
                        if (tab_goto[indi1] == tab_etiq[indi2])
                        {
                                sw_found = 1;
                        }
                }
                if (sw_found == 0)
                {
                        clrscr();puts("No se encuentra etiqueta.");exit(1);
                }
        }
        fcloseall();
        system(exec); system(exec2);
}
void compv(void)
{
    num_ent++;
    if (((swv >= 2) || (swo == 0)) && (num_ent > 1))
    {
        puts("Error semantico en expresion. Dos variables juntas. ");
        printf("%i",num_linea); exit(1);
    }
    swc = 0; swo = 0;
}
```

```
void compc(void)
{
    num_ent++;
    if (((swc >= 2) || (swo == 0)) && (num_ent > 1))
    {
        puts("Error semantico en expresion. Dos constantes juntas. ");
        printf("%i",num_linea); exit(1);
    }
    swv = 0; swo = 0;
}

void compo(void)
{
    num_ent++;
    if (((swo >= 2) || (swc == 0) && (swv == 0)) && (num_ent > 1))
    {
        puts("Error semantico en expresion. Dos operadores juntos. ");
        printf("%i",num_linea); exit(1);
    }
    swc = 0; swv = 0;
}

void error_o(void)
{
    if (swo != 0)
    {
        puts("Acaba en operador. "); printf("%i",num_linea); exit(1);
    }
    swc = 0; swv = 0; swo = 0; num_ent = 0;
}
```

Rutuinas de generacion de codigo

```
void gen_cls()
{
   strcpy(li_asm,"\n\tmov\tax,0700h");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tmov\tbh,07h");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tmov\tcx,0000h");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tmov\tdx,2479h");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tint\t10h\n");fputs(li_asm,file4);
}
```

```
void gen_gotoxy()
{
   itoa(primer_cons,uso,16);itoa(ult_cons,uso2,16);
   strcpy(li_asm,"\n\tmov\tah,02h\n");fputs(li_asm,file4);
   strcpy(li_asm,"\tmov\tbh,00h\n");fputs(li_asm,file4);
   strcpy(li_asm,"\tmov\tdx,");
   if(primer_cons < 16)
     strcat(li_asm,"0");
   strcat(li_asm,uso);
   if(ult_cons < 16)
     strcat(li_asm,"0");
   strcat(li_asm,uso2);
   strcat(li_asm,"h\n");fputs(li_asm,file4);
   strcpy(li_asm,"\tint\t10h\n");fputs(li_asm,file4);
}

void gen_read(char *arg1)
{
   char temp[16]="";
   if (tipo == 2)
   {
        strcpy(temp,arg1);
        strcpy(arg1,"n_car");
        strcpy(li_asm,"\n\tcall\tinic_r");fputs(li_asm,file4);
   }
   strcpy(li_asm,"\n\tmov\tah,03Fh\n");fputs(li_asm,file4);
   strcpy(li_asm,"\tmov\tbx,0000h\n");fputs(li_asm,file4);
   strcpy(li_asm,"\tmov\tcx,size ");strcat(li_asm,arg1);fputs(li_asm,file4);
   strcpy(li_asm,"\n\tadd\tcx,0002h");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tmov\tdx,offset ");strcat(li_asm,arg1);fputs(li_asm,file4);
   strcpy(li_asm,"\n\tint\t21h\n");fputs(li_asm,file4);
   if (tipo == 2)
   {
        strcpy(li_asm,"\n\tcall\tread_n"); fputs(li_asm,file4);
        strcpy(li_asm,"\n\tmov\tword ptr ax,ds:[n_car_]"); fputs(li_asm,file4);
        strcpy(li_asm,"\n\tmov\tword ptr ds:[");
        strcat(li_asm,temp); strcat(li_asm,"],ax\n");fputs(li_asm,file4);
   }
}
```

```
void gen_write(char *arg1)
{
   if (tipo == 2)
   {
        strcpy(li_asm,"\n\tmov\tax,ds:["); strcat(li_asm,arg1); strcat(li_asm,"]\n");fputs(li_asm,file4);
        strcpy(li_asm,"\tcall\twrite_n\n"); fputs(li_asm,file4);
   }
   else
   {
        strcpy(li_asm,"\n\tmov\tah,040h\n");fputs(li_asm,file4);
        strcpy(li_asm,"\tmov\tbx,0001h\n");fputs(li_asm,file4);
        strcpy(li_asm,"\tmov\tcx,size ");strcat(li_asm,arg1);fputs(li_asm,file4);
        strcpy(li_asm,"\n\tmov\tdx,offset ");strcat(li_asm,arg1);fputs(li_asm,file4);
        strcpy(li_asm,"\n\tint\t21h\n");fputs(li_asm,file4);
   }
}

void gen_write_lit()
{
   cont_lit += 1;itoa(cont_lit,uso,10);strcpy(uso2,"lit_");strcat(uso2,uso);
   strcpy(li_asm,"\n\tmov\tah,040h\n");fputs(li_asm,file4);
   strcpy(li_asm,"\tmov\tbx,0001h\n");fputs(li_asm,file4); n2 = 0;

   p_l2 = p_l1;
   for(n3 = 1; n3 != cont_lit; n3++)
   {
        p_l2 = p_l2->l1;
   }
   n2 = strlen(p_l2->c1) - 3;

   itoa(n2,uso,10);strcpy(li_asm,"\tmov\tcx,");
   strcat(li_asm,uso);strcat(li_asm,"d\n");fputs(li_asm,file4);
   strcpy(li_asm,"\tmov\tdx,offset ");fputs(li_asm,file4);
   strcpy(li_asm,uso2);fputs(li_asm,file4); strcpy(li_asm,"\n");fputs(li_asm,file4);
   strcpy(li_asm,"\tint\t21h\n");fputs(li_asm,file4);
}
```

```
void gen_write_n()
{
   strcpy(li_asm,"\nwrite_n\tproc near");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tcmp\tax,0000h");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tjge\twri1");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tnot\tax");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tadd\tax,01h");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tpush\tax");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tmov\tah,02h");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tmov\tdl,0C4h");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tint\t21h");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tpop\tax");fputs(li_asm,file4);
   strcpy(li_asm,"\nwri1:\tmov\tcx,10000d");fputs(li_asm,file4);
   strcpy(li_asm,"\nfuera0:\tpush\tcx");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tmov\tdx,0000h");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tcmp\tax,cx");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tjb\tfuera");fputs(li_asm,file4);
   strcpy(li_asm,"\ndentro:\tinc\tdx");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tsub\tax,cx");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tcmp\tax,cx");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tjb\tfuera");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tjmp\tdentro");fputs(li_asm,file4);
   strcpy(li_asm,"\nfuera:\tpop\tcx");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tpush\tax");fputs(li_asm,file4);
   strcpy(li_asm,"\n\txchg\tax,cx");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tcmp\tax,0001");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tje\tfuera1");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tmov\tbx,10d");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tpush\tdx");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tmov\tdx,0000h");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tdiv\tword ptr bx");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tpop\tdx");fputs(li_asm,file4);
   strcpy(li_asm,"\n\txchg\tax,cx");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tmov\tah,02h");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tadd\tdx,030h");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tint\t21h");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tpop\tax");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tjmp\tfuera0");fputs(li_asm,file4);
   strcpy(li_asm,"\nfuera1:");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tpop\tax");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tmov\tah,02h");fputs(li_asm,file4);
```

```
    strcpy(li_asm,"\n\tadd\tdx,030h");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tint\t21h");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tret");fputs(li_asm,file4);
    strcpy(li_asm,"\nwrite_n\tendp\n");fputs(li_asm,file4);
}

void gen_read_n()
{
    strcpy(li_asm,"\nread_n\tproc near");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tmov\tsi,offset n_car");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tadd\tsi,0005h");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tmov\tds:[n_car_],0");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tmov\tcx,0001h");fputs(li_asm,file4);
    strcpy(li_asm,"\naqui:\txor\tax,ax");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tmov\tbyte ptr al,ds:[si]");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tcmp\tal,02Dh");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tje\taqui3");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tcmp\tal,030h");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tjb\taqui2");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tcmp\tal,039h");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tja\taqui2");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tsub\tax,0030h");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tmul\tword ptr cx");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tadd\tds:[n_car_],ax");fputs(li_asm,file4);
    strcpy(li_asm,"\n\txchg\tax,cx");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tcmp\tsi,0000h");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tje\tfuera2");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tmov\tcx,0010d");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tmul\tword ptr cx");fputs(li_asm,file4);
    strcpy(li_asm,"\n\txchg\tax,cx");fputs(li_asm,file4);
    strcpy(li_asm,"\naqui2:\tcmp\tsi,0000h");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tje\tfuera2");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tdec\tsi");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tjmp\taqui");fputs(li_asm,file4);
    strcpy(li_asm,"\naqui3:\tmov\tax,0000h");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tsub\tax,ds:[n_car_]");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tmov\tds:[n_car_],ax");fputs(li_asm,file4);
    strcpy(li_asm,"\nfuera2:");fputs(li_asm,file4);
    strcpy(li_asm,"\n\tret");fputs(li_asm,file4);
    strcpy(li_asm,"\nread_n\tendp\n");fputs(li_asm,file4);
}
```

```
void gen_inic_r()
{
   strcpy(li_asm,"\ninic_r\tproc near");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tmov\tsi,offset n_car");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tmov\tword ptr ds:[si],\" \"");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tmov\tword ptr ds:[si+2],\" \"");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tmov\tword ptr ds:[si+4],\" \"");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tmov\tbyte ptr ds:[si+6],\" \"");fputs(li_asm,file4);
   strcpy(li_asm,"\n\tret");fputs(li_asm,file4);
   strcpy(li_asm,"\ninic_r\tendp\n\n");fputs(li_asm,file4);
}

void gen_for()
{
   cont_for += 1; tab_for[sw_for] = cont_for;
   itoa(cont_for,uso,10);strcpy(uso2,"f_");strcat(uso2,uso);
   itoa(primer_cons,uso3,16);itoa(ult_cons,uso4,16);
   strcpy(li_asm,"\n\tmov\tds:[");strcat(li_asm,ult_var);strcat(li_asm,"],");
   if(primer_cons < 16)
     strcat(li_asm,"0");
   strcat(li_asm,uso3);strcat(li_asm,"h\n");fputs(li_asm,file4);
   strcpy(li_asm,"\tmov\tcx,");
   if(primer_cons < 16)
     strcat(li_asm,"0");
   strcat(li_asm,uso3);strcat(li_asm,"h\n");fputs(li_asm,file4);
   strcpy(uso3,"f_j_");strcat(uso3,uso);strcat(uso3,":\t");
   strcpy(li_asm,uso3);strcat(li_asm,"cmp\tcx,");
   if(ult_cons < 16)
     strcat(li_asm,"0");
   strcat(li_asm,uso4);strcat(li_asm,"h\n");fputs(li_asm,file4);
   strcpy(uso3,"faf_");strcat(uso3,uso);
   strcpy(li_asm,"\tja\t");strcat(li_asm,uso3);strcat(li_asm,"\n");fputs(li_asm,file4);
   strcpy(li_asm,"\tinc\tcx\n");fputs(li_asm,file4);
   strcpy(li_asm,"\tpush\tcx\n");fputs(li_asm,file4);
   strcpy(uso3,"f_");strcat(uso3,uso);strcat(uso3,"__");
   strcpy(li_asm,"\tcall\t");strcat(li_asm,uso3);strcat(li_asm,"\n");fputs(li_asm,file4);
   strcpy(li_asm,"\tinc\tds:[");strcat(li_asm,ult_var);strcat(li_asm,"]\n");fputs(li_asm,file4);
   strcpy(li_asm,"\tpop\tcx\n");fputs(li_asm,file4);
   strcpy(uso3,"f_j_");strcat(uso3,uso);
   strcpy(li_asm,"\tjmp\t ");strcat(li_asm,uso3);strcat(li_asm,"\n");fputs(li_asm,file4);
   strcpy(uso3,"faf_");strcat(uso3,uso);
```

```
    strcpy(li_asm,uso3);strcat(li_asm,":\tjmp\t");
    strcpy(uso3,"f_f_");strcat(uso3,uso);
    strcat(li_asm,uso3);strcat(li_asm,"\n\n");fputs(li_asm,file4);
    strcpy(uso3,"f_");strcat(uso3,uso);strcat(uso3,"__");
    strcpy(li_asm,uso3);strcat(li_asm,"\tproc\tnear");fputs(li_asm,file4);
}

void gen_for2()
{
    itoa(tab_for[sw_for+1],uso,10);strcpy(uso2,"f_");
    strcat(uso2,uso);strcat(uso2,"__\tendp\n");
    strcpy(li_asm,"\n\tret\n");fputs(li_asm,file4);
    strcpy(li_asm,uso2);fputs(li_asm,file4);
    strcpy(li_asm,"f_f_");strcat(li_asm,uso);strcat(li_asm,":\n");fputs(li_asm,file4);
}

void gen_if0()
{
    cont_if += 1;itoa(cont_if,uso,10);strcpy(uso2,"c_");strcat(uso2,uso);
}

void gen_if()
{
    p_tok = strtok(li_polaca," ");
    while (p_tok != NULL)
    {
        if ((*p_tok == '6') || (*p_tok == '7'))
        {
          s_p_tok = 1;
          strcpy(li_asm,"\n\tpop\tax"); fputs(li_asm,file4);
          strcpy(li_asm,"\n\tpop\tbx"); fputs(li_asm,file4);
          switch(*p_tok)
          {
                case '6': strcpy(li_asm,"\n\tand\tax,bx"); fputs(li_asm,file4); break;
                case '7': strcpy(li_asm,"\n\tor\tax,bx"); fputs(li_asm,file4); break;
          }
          strcpy(li_asm,"\n\tpush\tax\n"); fputs(li_asm,file4);
          p_tok = strtok(NULL," ");
          if (p_tok == NULL)
          {
             itoa(tab_w_if[sw_while + sw_if],uso,10);strcpy(uso2,"cond_");strcat(uso2,uso);
```

```
            strcpy(li_asm,"\n\tpop\tax"); fputs(li_asm,file4);
            strcpy(li_asm,"\n\tcmp\tax,0001h"); fputs(li_asm,file4);
            strcpy(li_asm,"\n\tje\t"); strcat(li_asm,uso2); strcat(li_asm,"a\n");
            fputs(li_asm,file4);
            strcpy(li_asm,"\n\tjmp\t"); strcat(li_asm,uso2); strcat(li_asm,"\n");
            fputs(li_asm,file4);
            strcpy(li_asm,uso2); strcat(li_asm,"a:\n"); fputs(li_asm,file4);
        }
    }
    else
    {
            strcpy(li_asm,"\n\tmov\tax,ds:[");strcat(li_asm,p_tok);
            strcat(li_asm,"]");fputs(li_asm,file4);
            p_tok = strtok(NULL," "); strcpy(li_asm,"\n\tmov\tbx,ds:[");
            strcat(li_asm,p_tok);strcat(li_asm,"]");fputs(li_asm,file4);
            strcpy(li_asm,"\n\tcmp\tax,bx"); fputs(li_asm,file4);
            p_tok = strtok(NULL," ");
      switch(*p_tok)
      {
            case '1': gen_if0(); strcpy(li_asm,"\n\tjb\t"); strcat(li_asm,uso2);
                  fputs(li_asm,file4); break;
            case '2': gen_if0(); strcpy(li_asm,"\n\tja\t"); strcat(li_asm,uso2);
                  fputs(li_asm,file4); break;
            case '3': gen_if0(); strcpy(li_asm,"\n\tjae\t"); strcat(li_asm,uso2);
                  fputs(li_asm,file4); break;
            case '4': gen_if0(); strcpy(li_asm,"\n\tjbe\t"); strcat(li_asm,uso2);
                   fputs(li_asm,file4); break;
            case '5': gen_if0(); strcpy(li_asm,"\n\tje\t"); strcat(li_asm,uso2);
                   fputs(li_asm,file4); break;
      }
      strcpy(li_asm,"\n\tpush\t0"); fputs(li_asm,file4);
      strcpy(li_asm,"\n\tjmp\t"); strcat(li_asm,uso2); strcat(li_asm,"_"); fputs(li_asm,file4);
      strcpy(li_asm,"\n"); strcat(li_asm,uso2); strcat(li_asm,":\t");
      strcat(li_asm,"push 1"); fputs(li_asm,file4);
      strcpy(li_asm,"\n"); strcat(li_asm,uso2); strcat(li_asm,"_:"); fputs(li_asm,file4);
      p_tok = strtok(NULL," ");
    }
}
if (s_p_tok == 0)
{
    strcpy(li_asm,"\n\tpop\tax"); fputs(li_asm,file4);
```

```
        itoa(tab_w_if[sw_while + sw_if],uso,10);strcpy(uso2,"cond_");strcat(uso2,uso);
        strcpy(li_asm,"\n\n\tcmp\tax,0001h"); fputs(li_asm,file4);
        strcpy(li_asm,"\n\tje\t"); strcat(li_asm,uso2); strcat(li_asm,"a\n"); fputs(li_asm,file4);
        strcpy(li_asm,"\n\tjmp\t"); strcat(li_asm,uso2); strcat(li_asm,"\n"); fputs(li_asm,file4);
        strcpy(li_asm,"\n"); strcat(li_asm,uso2); strcat(li_asm,"a:\n"); fputs(li_asm,file4);
    }
}

void gen_asig()
{
    comp_var(bak_var);
    if (tipo == 1)
    {
        clrscr();
        printf("%s %d","Error semantico. Numero a variable caracter. ",num_linea);
        exit(1);
    }
    p_tok = strtok(li_polaca," ");
    while (p_tok != NULL)
    {
        if ((*p_tok == '+') || (*p_tok == '-') || (*p_tok == '*') || (*p_tok == '/'))
        {
          strcpy(li_asm,"\n\tpop\tbx"); fputs(li_asm,file4);
          strcpy(li_asm,"\n\tpop\tax"); fputs(li_asm,file4);
          switch(*p_tok)
          {
                case '+': strcpy(li_asm,"\n\tadd\tax,bx"); fputs(li_asm,file4); break;
                case '-': strcpy(li_asm,"\n\tsub\tax,bx"); fputs(li_asm,file4); break;
                case '*': strcpy(li_asm,"\n\tmul\tbx"); fputs(li_asm,file4); break;
                case '/': strcpy(li_asm,"\n\tpush\tdx"); fputs(li_asm,file4);
                          strcpy(li_asm,"\n\tmov\tdx,0000h"); fputs(li_asm,file4);
                          strcpy(li_asm,"\n\tdiv\tword ptr bx"); fputs(li_asm,file4);
                          strcpy(li_asm,"\n\tpop\tdx"); fputs(li_asm,file4); break;
          }
          strcpy(li_asm,"\n\tpush\tax"); fputs(li_asm,file4);
        }
        else
        {
          if ((*p_tok >= '0') && (*p_tok <= '9'))
          {
                strcpy(li_asm,"\n\tpush\t");strcat(li_asm,p_tok);fputs(li_asm,file4);
```

```
            }
            else
            {
                  strcpy(li_asm,"\n\tpush\tds:[");strcat(li_asm,p_tok);
                  strcat(li_asm,"]");fputs(li_asm,file4);
            }
         }
         p_tok = strtok(NULL," ");
    }
    strcpy(li_asm,"\n\tpop\tax"); fputs(li_asm,file4);
    strcpy(li_asm,"\n\tmov\tds:[");strcat(li_asm,bak_var);strcat(li_asm,"],ax\n");fputs(li_asm,file4);
}

void gen_asig2()
{
   comp_var(bak_var);
   if (tipo == 2)
   {
         clrscr();
         printf("%s %d","Error semantico. Caracter a variable numerica. ",num_linea);
         exit(1);
   }
   cont_lit += 1;itoa(cont_lit,uso,10);strcpy(uso2,"lit_");strcat(uso2,uso);
   strcpy(li_asm,"\tpush\tds\n");fputs(li_asm,file4);
   strcpy(li_asm,"\tpop\tes\n");fputs(li_asm,file4);
   strcpy(li_asm,"\tmov\tsi,offset ");strcat(li_asm,uso2);strcat(li_asm,"\n");fputs(li_asm,file4);
   strcpy(li_asm,"\tmov\tdi,offset ");strcat(li_asm,ult_var);
   strcat(li_asm,"\n");fputs(li_asm,file4);
   if ((pun_c2 - pun_c1) > longi)
   {
[illegible]
   }
   else
   {
     itoa(pun_c2 - pun_c1,uso,10);
   }
   strcpy(li_asm,"\tmov\tcx, ");strcat(li_asm,uso);strcat(li_asm,"d\n");fputs(li_asm,file4);
   strcpy(li_asm,"\trepe\tmovsb\n");fputs(li_asm,file4);
}
```

```
void gen_etiq()
{
    itoa(ult_cons,uso,10);strcpy(uso2,"\netiq_");strcat(uso2,uso);
    strcpy(li_asm,uso2);strcat(li_asm,":\n");fputs(li_asm,file4);
}

void gen_goto()
{
    itoa(ult_cons,uso,10);strcpy(uso2,"etiq_");strcat(uso2,uso);
    strcpy(li_asm,"\n\tjmp\t");strcat(li_asm,uso2);strcat(li_asm,"\n");fputs(li_asm,file4);
}

void polaca(void)
{
    strcat(li_polaca,ant_var); strcat(li_polaca," "); strcat(li_polaca,ult_var); strcat(li_polaca," ");
    tabla_ope[ind_p] = tok_oper; ind_p += 1;
}

void polaca1(void)
{
    ind_p-=1;
    while ((tabla_ope[ind_p] == '1') || (tabla_ope[ind_p] == '2') ||
        (tabla_ope[ind_p]== '3') || (tabla_ope[ind_p] == '4') ||
        (tabla_ope[ind_p] == '5'))
    {
        strncat(li_polaca,&tabla_ope[ind_p],1); strcat(li_polaca," "); ind_p-=1;
    }
    ind_p+=1; tabla_ope[ind_p] = tok_oper; ind_p+=1;
}

void polaca_comun()
{
    ind_p-=1;
    while ((tabla_ope[ind_p] != '(') && (ind_p >= 0))
    {
        strncat(li_polaca,&tabla_ope[ind_p],1); strcat(li_polaca," "); ind_p-=1;
    }
}
```

```
void polaca_exp1(void)
{
    strcat(li_polaca,ult_var); strcat(li_polaca," ");
}

void polaca_exp2(void)
{
    strcat(li_polaca,itoa(ult_cons,uso,10)); strcat(li_polaca," ");
}

void polaca_exp3(void)
{
    if (ind_p == 0)
    {
        tabla_ope[ind_p] = tok_oper; ind_p += 1;
    }
    else
    {
        ind_p-=1;
        while ((tabla_ope[ind_p] == '*') || (tabla_ope[ind_p] == '/'))
        {
          strncat(li_polaca,&tabla_ope[ind_p],1); strcat(li_polaca," "); ind_p-=1;
        }
        ind_p+=1; tabla_ope[ind_p] = tok_oper; ind_p+=1;
    }
}

void limpiar_polaca(void)
{
    sw_if2 = 1; ind_p = 0; strcpy(li_polaca,""); strcpy(tabla_ope,"");
}

yylex(void)
{
        if (sw_yy == 0)
        {
                fich_for[strlen(fich_for)-3] = '\0'; strcat(fich_for,"lex");
                sw_yy = 1;
                if ((file3 = fopen(fich_for,"rt")) == NULL)
                {
                        clrscr();gotoxy(1,22);
```

```
                printf("Imposible abrir fichero %s",fich_for);exit(1);
        }
        fgets(li_tok2,200,file3);pun = li_tok2;num_linea = 1;
}

if (strncmp(pun," ",1) == 0)
{
        pun++;
}
pun_c1 = pun;
for(; (strncmp(pun," ",1) != 0 && strncmp(pun,"\n",1) != 0); pun++);
pun_c2 = pun;
strcpy(tok_res,"                ");
strncpy(tok_res,pun_c1,pun_c2-pun_c1);

if ((strncmp(pun,"\n",1) == 0) && (pun_c2-pun_c1 == 0))
{

        if (sw_L == 1)
        {
            sw_L = 0; sw_primer_cons = 0;
            comp_for(primer_cons,ult_cons);
            gen_for();
        }
        if (sw_O == 1)
        {
            sw_O = 0; cont_while += 1; tab_while[sw_while] = cont_while;
            itoa(tab_while[sw_while],uso,10);strcpy(uso2,"\nc_w_");strcat(uso2,uso);
            strcpy(li_asm,uso2); strcat(li_asm,":"); fputs(li_asm,file4);
            sw_if2 = 0; s_p_tok = 0;
            gen_if(); limpiar_polaca();
        }
        fgets(li_tok2,200,file3);pun = li_tok2;num_linea += 1;
        gotoxy(38,18);printf("%i",num_linea);
        return ('\n');
}

if (strncmp(tok_res,"var",3) == 0)
{
        indice1 = 0;
        strcpy(ant_var,"\x0\x0\x0\x0\x0\x0\x0\x0\x0\x0\x0\x0\x0");
```

```
        strcpy(ant_var,ult_var);
        strcpy(ult_var,"");
        for(; (((tok_res[indice1] >= 48) && (tok_res[indice1] <= 57)) ||
        ((tok_res[indice1] >= 97) && (tok_res[indice1] <= 122))) ; indice1++);
        indice1++;
        strcat(ult_var,&tok_res[indice1]);
        indice1 = 0;
        for(; (((ult_var[indice1] >= 48) && (ult_var[indice1] <= 57)) ||
        ((ult_var[indice1] >= 97) && (ult_var[indice1] <= 122))) ; indice1++);
        ult_var[indice1] = '\x0';
        return var;
}

if (strncmp(tok_res,"lit",3) == 0)
{
        pun++; pun++;
        pun_c1 = pun;
        for(; (strncmp(pun,"\"",1) != 0); pun++);
        pun_c2 = pun;
        strcpy(tok_res,"                ");
        strncpy(tok_res,pun_c1,pun_c2-pun_c1);

        fgets(li_tok2,200,file3);pun = li_tok2;num_linea += 1;
        gotoxy(38,18);printf("%i",num_linea);
        if (feof(file3))
        {
                fcloseall();exit(1);
        }
        return lit;
}

if (strncmp(tok_res,"cons",4) == 0)
{
        strcpy(tok_res2,"");indice1 = 0;
        for(; (((tok_res[indice1] >= 48) && (tok_res[indice1] <= 57)) ||
        ((tok_res[indice1] >= 97) && (tok_res[indice1] <= 122))) ; indice1++);
        indice1++;
        strcat(tok_res2,&tok_res[indice1]);
        indice1 = 0;
        for(; (((tok_res2[indice1] >= 48) && (tok_res2[indice1] <= 57)) ||
         ((tok_res2[indice1] >= 97) && (tok_res2[indice1] <= 122))) ; indice1++);
```

```
        tok_res2[indice1] = '\x0';
        if (sw_primer_cons == 0)
        {
                sw_primer_cons = 1;
                primer_cons = atoi(tok_res2);
                ult_cons = atoi(tok_res2);
        }
        else
        {
                ult_cons = atoi(tok_res2);
        }
        return cons;
}

if (strncmp(tok_res,">=",2) == 0)
{
        return 'a';
}
if (strncmp(tok_res,"<=",2) == 0)
{
        return 'b';
}

switch(*tok_res)
{
        case '=': strcpy(bak_var,ult_var); break;
        case 'G': cont_cond += 1;sw_if += 1;
                tab_w_if[sw_while + sw_if] = cont_cond;
                sw_if2 = 0; s_p_tok = 0; gen_if(); limpiar_polaca(); break;
        case 'H': itoa(tab_w_if[sw_while + sw_if],uso,10);
                strcpy(uso2,"cond_");strcat(uso2,uso);
                strcpy(li_asm,"\n\tjmp\t"); strcat(li_asm,uso2);
                strcat(li_asm,"_\n");fputs(li_asm,file4);
                strcpy(li_asm,uso2); strcat(li_asm,":\n"); fputs(li_asm,file4);
                tabla_h[sw_if] = 1;
                break;
        case 'I': itoa(tab_w_if[sw_while + sw_if],uso,10);
                strcpy(uso2,"cond_");strcat(uso2,uso);
                if (tabla_h[sw_if] == 0)
                {
                  strcpy(li_asm,uso2); strcat(li_asm,":\n"); fputs(li_asm,file4);
```

```
            }
            else
            {
              strcpy(li_asm,uso2); strcat(li_asm,"_:"); fputs(li_asm,file4);
            }
            tabla_h[sw_if] = 0; sw_if -= 1;break;
case 'J': sw_for += 1; sw_primer_cons = 0; break;
case 'M': sw_for -= 1; break;
case 'N': cont_cond += 1;sw_while += 1;
             tab_w_if[sw_while + sw_if] = cont_cond; sw_O = 1;
             limpiar_polaca();  break;
case 'O': itoa(tab_while[sw_while],uso,10);
            strcpy(uso2,"c_w_");strcat(uso2,uso);
            strcpy(li_asm,"\n\tjmp\t"); strcat(li_asm,uso2);
            strcat(li_asm,"\n");fputs(li_asm,file4);
            itoa(tab_w_if[sw_while + sw_if],uso,10);
            strcpy(uso2,"cond_");strcat(uso2,uso);
            strcpy(li_asm,uso2); strcat(li_asm,":\n"); fputs(li_asm,file4);
            sw_while -= 1; break;
case 'K': sw_L = 1; break;
case 'F': limpiar_polaca(); break;
case '(': sw_sema_cond = 0;
          if(sw_if2 == 1) tabla_ope[ind_p]='('; ind_p+=1; break;


case ')': if(sw_if2 == 1)
            {
              polaca_comun();
            }
            break;
case '}' : if (sw_ends == 0)
             {
               sw_ends = 1;
               strcpy(li_asm,"data\tends\n\n");fputs(li_asm,file4);
               strcpy(li_asm,"codigo\tsegment\n");fputs(li_asm,file4);
               strcpy(li_asm,"\tassume cs:codigo,ds:data,ss:pila\n");
                  fputs(li_asm,file4);
               strcpy(li_asm,nom_prog);strcat(li_asm,"\tproc far\n");
                  fputs(li_asm,file4);
               strcpy(li_asm,"\tmov\tax,data\n");fputs(li_asm,file4);
               strcpy(li_asm,"\tmov\tds,ax\n");fputs(li_asm,file4);
```

```
                    }
                    else
                    {
                      strcpy(li_asm,"\n");fputs(li_asm,file4);
                      strcpy(li_asm,"\tmov\tax,4C00h\n");fputs(li_asm,file4);
                      strcpy(li_asm,"\tint\t21h\n");fputs(li_asm,file4);
                      strcpy(li_asm,nom_prog);strcat(li_asm,"\tendp\n");
                         fputs(li_asm,file4);
                      gen_write_n();
                      gen_read_n();
                      gen_inic_r();
                      strcpy(li_asm,"codigo ends\n");fputs(li_asm,file4);
                      strcpy(li_asm,"end\t");strcat(li_asm,nom_prog);fputs(li_asm,file4);
                    }
        }
        return (*tok_res);
}
```