Universidad Autónoma de Querétaro
Facultad de Ingeniería
Maestría en Instrumentación y Control Automático

Desarrollo de la Ingeniería de Software para la Implementación de un CNC

TESIS

Que como parte de los requisitos para obtener el grado de

Maestro en Ciencias en Instrumentación y Control Automático

Presenta:
Ing. Aurora Femat Díaz

Dirigido por:
Dr. Gilberto Herrera Ruiz

SINODALES

Dr. Gilberto Herrera Ruiz
Presidente — Firma

M. C. Pedro Daniel Alaniz Lumbreras
Secretario — Firma

M. C. Rodrigo Castañeda Miranda
Vocal — Firma

M. C. Roberto Augusto Gómez Loenzo
Suplente — Firma

M. C. Agustín Bravo Curiel
Suplente — Firma

M. I. Gerardo René Serrano Gutiérrez
Director de la Facultad

Dr. Sergio Quesada Aldana
Director de Investigación y Posgrado

Centro Universitario
Querétaro, Qro.
Noviembre de 2004
México

Resumen

El objetivo de la elaboración de esta tesis es la ingeniería de software para crear un sistema de desarrollo de programas CNC para máquinas-herramienta. El desarrollo contiene un editor especializado, un compilador y un módulo de pruebas de directivas, que corren en PC sobre sistema operativo Windows; se describe la parte de ejecución del lenguaje de control numérico así como los resultados del mismo. El editor permite escribir código de entrada según lo establecido por el estándar ISO, el compilador genera como salida código utilizado por la tarjeta de movimiento Galil así como registros para la activación o desactivación de los diferentes dispositivos que se pueden manejar a través de programa. La innovación del trabajo consiste en utilizar un compilador en lugar de los intérpretes convencionales utilizados por la mayoría de los CNC actuales, con el objetivo de realizar todas aquellas validaciones que se consideren necesarias para los movimientos de corte o posicionamiento de cada uno de los ejes en la máquina-herramienta o para los mecanismos que la acompañan, esto gracias a las diferentes etapas del proceso de compilación. Además el sistema está diseñado de manera flexible para poder ser adaptado a diferentes máquinas-herramientas estableciendo un método en el que los datos para las validaciones pueden ser modificados a través del mismo sistema. El CNC al que se integrará el compilador propuesto está formado por una computadora personal estándar, la tarjeta del controlador y las interfases para adaptarse a la máquina-herramienta. En el primer capítulo se introduce al tema de control numérico computarizado y máquina-herramienta. El segundo explica las bases de teoría de compiladores. El tercer capítulo describe los fundamentos para poder desarrollar programas para maquina-herramienta y la estructura de las principales instrucciones. El cuarto capítulo describe detalladamente como se realizó el desarrollo de este trabajo y las técnicas empleadas. Finalmente el quinto capítulo trata de las pruebas realizadas con una máquina-herramienta realizando maquinados y las conclusiones.

(Palabras clave: CNC, Compilador, Máquina-Herramienta)

Abstract

The goal of this thesis is the software engineering to create a development system of CNC programs for machine tools. The development contains a specialized editor, a compiler and a module for directives testing, which run on the Windows operating system; the CNC language execution and its results are also described. The editor allows the writing of the input code as defined by the ISO standard; the compiler generates the output code for Galil controller board and on/off registers for the machine devices that might be controlled by a CNC program. The innovation of this work is to use a compiler instead of conventional interpreters used with current CNC systems in order to perform anticipated validations appropriate for cutting work directives, or machine tool axis movements, or support devices start/stop actions, these taking advantages of each one of the compilation steps. Furthermore, the system has a flexible design to be adapted and extended to be used with different machine tools establishing a method in which validation data can be modified through the same system. The compiler's target CNC is integrated by a standard personal computer, the controller board, and I/O machine tool interfaces. Chapter one is an introduction to numeric control and machine tools. Chapter two explains compilers theory elements. Chapter three describes machine tools CNC programming fundamentals and main commands structure. Chapter four details the software case development and applied methods. Finally chapter five elaborates on the system testing with an actual machining job execution and conclusions.

(Keywords: CNC, compiler, machine tool)

Agradecimientos

A Dios por su amor y su cuidado en todo momento para conmigo y mi familia.

Al Dr. Gilberto Herrera por su apoyo y su ánimo en cada momento, sin su ayuda no lo hubiera logrado, gracias Doctor.

A Pao por su amor y su ternura.

A Eric por su apoyo y compañía constantes.

A mi mamá y mis hermanos por su amor y su comprensión.

Al Dr. Victor Hernández por sus enseñanzas y por compartir sus conocimientos conmigo cada vez que necesité ayuda durante el estudio de la Maestría.

A mis profesores y compañeros por todo lo que aprendí de ustedes, muchas gracias.

A mis hermanos por sus oraciones.

Al CONACYT por el apoyo económico.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Antecedentes

1.1. Introducción

1.1.1. Máquinas-herramienta con CNC:

El término *máquina-herramienta* abarca un amplio rango de equipos usados para el proceso de piezas fabricadas por medio de operaciones de arranque de viruta o remoción electroquímica de material.

Herrera en [Herrera, 1992] escribe "Una *máquina-herramienta* es una herramienta motorizada, que no es portátil durante su operación, usada para efectuar individualmente o en combinación, las operaciones de maquinado, formado y procesamiento electroquímico de metales, madera, vidrio, plástico y materiales similares".

El concepto *control numérico* se refiere a la unidad de control solamente, o a todas las partes que se integran en una solución de automatización de una máquina-herramienta u otros procesos a los que se les ha incorporado esta tecnología. Dos definiciones del estándar ANSI Z94.0 dadas en IEEE 1991 son:

1. Una técnica para controlar acciones de una máquina-herramienta y equipo similar por la inserción directa de información numérica en un punto dado. La información es automáticamente interpretada.
2. Cualquier sistema de control, más el equipo controlado que acepta comandos, información e instrucciones en forma simbólica como una entrada y convierte esta

información en una salida física y en valores físicos tales como dimensiones y cantidades.

También en [Herrera, 1992] se comenta que se agrega el término computarizado para indicar que se usa una computadora como medio para procesar la información numérica de entrada y convertirla en una salida apropiada para ser manejada por los controladores y actuadores del sistema, sustituyendo así mediante software tanto hardware como sea posible.

En el estándar ANSI Z94.0 en IEEE 1991 tenemos que control numérico computarizado se define como: "Un sistema de control numérico auto-contenido para una sola máquina-herramienta utilizando una computadora dedicada controlada por instrucciones almacenadas para desempeñar algunas o todas las funciones básicas de control numérico. Cintas perforadas y grabadoras de cinta no son usadas excepto, posiblemente, como respaldo en el evento de falla de la computadora. A través de conexión directa a un procesador central, el sistema CNC puede ser parte de un sistema de control numérico directo (DNC)".

El control numérico como alternativa técnicamente factible para la automatización de máquinas-herramienta se desarrolló entre 1949 y 1952 por Parsons Corporation y el Massachusetts Institute of Technology, bajo los auspicios de la U.S. Air Force [Seames, 1990].

La primera generación usó circuitos digitales, sin tener realmente una unidad central de procesamiento, esta generación se distingue por una lógica fija permanentemente conectada en sus circuitos y se refiere como control numérico NC o de conexión permanente (hardwired). En 1970 se desarrollaron las primeras máquinas-herramienta que incorporaban minicomputadoras como unidades de control, es decir control numérico computarizado (CNC). Con el desarrollo reciente de la electrónica y la tecnología de computación, las máquinas-herramienta modernas incorporan varios microprocesadores y controladores lógicos programables que comparten las tareas de control [Altintas, 2000]. Aun cuando los términos son intercambiables en la práctica actual, debe entenderse que el término CNC no puede aplicarse a los sistemas anteriores a 1970 con lógica fija.

Una máquina-herramienta CNC tiene tres unidades fundamentales:

1. La máquina-herramienta como tal, es decir, los elementos que permiten realizar el trabajo mecánico de arranque de viruta o desbaste, sujeción de las piezas de trabajo, los buriles y cortadores y la transmisión mecánica de potencia;

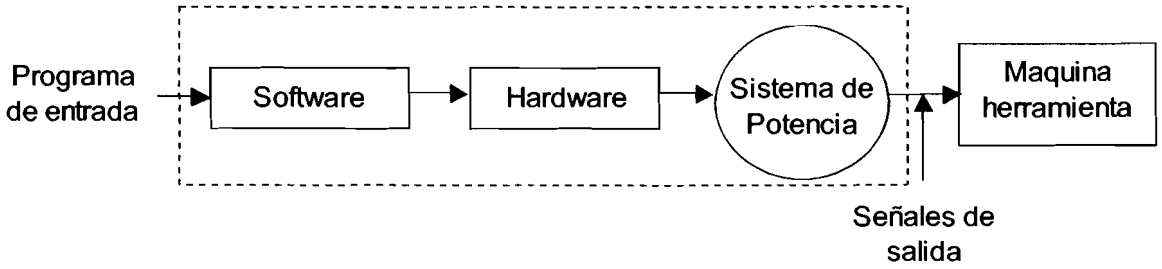


Figura 1.1: Diagrama esquemático de un sistema CNC

2. unidades de potencia como motores eléctricos y amplificadores y

3. la unidad de CNC propiamente.

Las unidades modernas de CNC incluyen varios CPUs de acuerdo al número de tareas de control y cálculo involucradas en los procesos cubiertos por el CNC como los ejes de movimiento, control de contorno, control de bancada, avances de herramienta, velocidades de corte, cambios de herramientas, fluido de corte, etc. Cada unidad CNC normalmente dispone de una computadora dedicada con sus dispositivos de entrada-salida y almacenamiento. Es frecuente incluso que en fábricas con varias máquinas-herramienta CNC estas se conecten en red, con un servidor central que puede disponer también del software apropiado para compartir la información del sistema MRP (Planeación de requerimientos de manufactura) de la compañía y entonces administrar la carga y descarga de programas de maquinado de acuerdo a la programación de la producción, tales sistemas reciben el nombre de sistemas distribuidos controlados numéricamente DNC [Altintas, 2000].

Es de particular importancia el hecho de que los costos de adquisición, instalación y mantenimiento de sistemas CNC son generalmente altos, por lo tanto su uso se restringe a las aplicaciones con volúmenes de producción que los justifiquen en tiempos razonables. La excepción puede ser su uso en laboratorios CAM para el diseño y fabricación rápida de prototipos o moldes de inyección de plástico, moldes para fundición por inyección, o aplicaciones semejantes, donde las reducciones en el tiempo de desarrollo de nuevos productos bien puede justificar la inversión en sistemas CAD / CAM basados en máquinas-herramienta CNC, a pesar de los bajos tiempos de utilización.

El CNC en máquinas-herramienta aumenta la precisión de las trayectorias de corte y su reproducibilidad repercutiendo en mayor estabilidad y mejor centrado del proceso de maquinado de las piezas, resultando en procesos más hábiles y piezas de mejor calidad

dada la selección apropiada de buriles y cortadores, reemplazo de los mismos y mantenimiento de la máquina y de la unidad CNC.

1.1.2. Características del software para CNC

El software de un sistema CNC debe cubrir normalmente tres programas: De la pieza, de servicio y de control. El programa de la pieza a maquinar consiste en la descripción de la geometría requerida y las condiciones de corte tales como velocidad del husillo y de alimentación. El programa de servicio es el que sirve para editar, verificar y corregir el programa de la pieza. El programa de control acepta el programa de la pieza como entrada y produce señales para dirigir los ejes de movimiento.

Los primeros dos módulos, es decir el programa de la pieza y el programa de servicio, funcionan asíncronos con el hardware de control. El programa de control funciona en tiempo real con el hardware para controlar la operación de maquinado.

En el presente trabajo se desarrollaron los tres módulos de la siguiente forma: El Editor es el programa de servicio que sirve para modificar y mantener el programa de la pieza que se escribe en código G. El Compilador convierte el código G de un programa completo en una secuencia de señales de control que servirán para operar el hardware de control. Por lo tanto esta secuencia de señales se constituye en el programa de control cuando funciona con el sistema CNC.

Hay un Verificador en línea que revisa el conjunto de operaciones de una línea y genera el código resultante para ser utilizado por el hardware de control, este verificador funciona juntamente con este hardware y su función ayudar a verificar que cada una de las instrucciones funcione de la manera esperada.

1.1.3. Tecnología de CNC disponible actualmente

Las máquinas-herramienta que operan con CNC abarcan la gran variedad de operaciones que se realizan de manera convencional, estas incluyen tornos, fresas, centros de maquinado, máquinas de electro-erosión de hilo o de electrodo, rectificadoras, cepillos, taladros, etc. Existen en el mercado tanto maquinaria ya automatizada con CNC, como sistemas CNC para ser instalados en maquinaria existente en acciones de modernización. Para amplio detalle en el proceso de modernización de máquinas-herramienta leer los capítulos 2 y 3 de [Herrera, 1992] donde Herrera desarrolla el concepto y para consultar el

caso de un servo sistema de una máquina fresadora CNC ver [Alaniz, 2003] que ejemplifica la modernización de máquinas-herramienta empleando el CNC CHROM-II.

La oferta de este tipo de sistemas, así como de los servicios relacionados como su mantenimiento, integración de sistemas CNC, modernización, unidades de control CNC, sensores, actuadores, entrenamiento y capacitación en CNC es muy amplia. Cada fabricante mantiene una gama en su oferta que cubre cierto número de las aplicaciones mencionadas, con tendencia a enfocarse en los mercados más rentables que normalmente atienden las necesidades de empresas con plantas grandes que son las que consumen este tipo de tecnología.

El tamaño de la planta es el factor dominante en la predicción de la adopción de CAD y CNC. En el sector industrial de México, solo el 2.1 % de las empresas son grandes o medianas (con más de 50 trabajadores) y el 97.9 % son pequeñas o micros (hasta 50 trabajadores), de acuerdo al Censo Económico 1999. Existen aún un número importante de instalaciones de maquinado que emplean equipos manuales, debido al alto costo de automatización, lo que significa que la competitividad por productividad y calidad de la planta productiva nacional en el ámbito de los maquinados está en desventaja por tecnología CNC y es una área de oportunidad significativa para el desarrollo tecnológico en el ámbito de la instrumentación y control.

1.1.4. Justificación del sistema de programación propuesto

Aunque la oferta de los productos CNC disponibles en el mercado cubre la amplia gama de soluciones para la industria, esta se encuentra enfocada a grandes plantas, por lo que el presente trabajo proporciona una opción en la pequeña y mediana empresa mexicanas como solución estándar al usar código G como entrada. Además este desarrollo se incorporará a hardware ya disponible, como parte de la unidad CNC CHROM-II citada en [Alaniz, 2003] para la modernización de máquinas-herramienta.

Capítulo 2

Teoría de Compiladores

2.1. Modelo de análisis y síntesis de la compilación

Aho en [Aho, Sethi y Ullman, 1986] describe un compilador como un programa que lee un programa escrito en un lenguaje, denominado lenguaje fuente, y lo traduce a un programa equivalente en otro lenguaje, conocido como lenguaje objeto .

El lenguaje objeto puede ser otro lenguaje de programación o el lenguaje máquina de cualquier computadora, donde esta puede variar desde un microprocesador hasta varias computadoras conectadas entre sí.

En la compilación hay dos partes: análisis y síntesis. La parte de análisis divide al programa fuente en sus elementos componentes y crea una representación intermedia del programa fuente. La parte de síntesis construye el programa objeto deseado a partir de la representación intermedia.

2.1.1. Las fases de un compilador

Conceptualmente, un compilador opera en fases, cada una de las cuales transforma al programa fuente de una representación a otra. En la práctica , se pueden agrupar algunas fases y las representaciones intermedias entre las fases no necesitan ser construidas explícitamente. En la figura 2.1 se muestran gráficamente las fases de un compilador.

La administración de la tabla de símbolos y el manejo de errores interactúan con cada

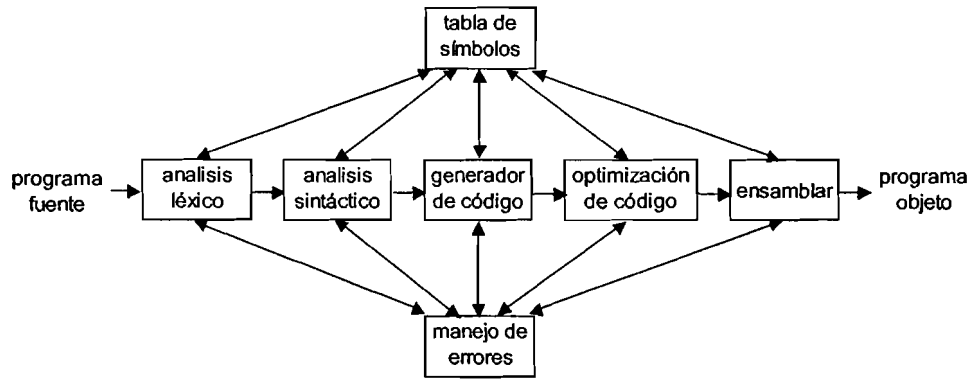


Figura 2.1: Fases de un compilador.

una de las fases.

2.1.2. Análisis del programa fuente

En los lenguajes de programación el análisis debe constar, al menos, de las siguientes dos fases:

1. Análisis lineal o análisis léxico, en el que las cadenas de caracteres que construyen el programa fuente se leen de izquierda a derecha y se agrupan en componentes léxicos, que son secuencias de caracteres que tienen un significado colectivo.

2. Análisis jerárquico o análisis sintáctico, en el que los componentes léxicos se agrupan jerárquicamente en colecciones anidadas con un significado colectivo.

Existe una tercer fase, llamada análisis semántico, que no siempre es construida explícitamente, cuando si lo es, sigue a la fase de análisis sintáctico; también es llamada verificación de tipos, por que garantiza la detección y comunicación, principalmente, de esa clase de errores; un error de tipo es, por ejemplo, cuando se aplica un operador a uno o más operandos incompatibles.

2.1.3. Síntesis del programa objeto

1. La fase de generación de código traduce cada una de las estructuras o instrucciones intermedias obtenidas de la fase de análisis a una secuencia de instrucciones de

máquina que ejecuta la misma tarea.

2. La fase de optimización de código trata de mejorar el código intermedio, de modo que resulte un código de máquina más rápido de ejecutar. Algunas optimizaciones no son triviales.

3. La fase final de un compilador es ensamblar; aquí se une el código objeto, obtenido de las fases anteriores, con los elementos necesarios: bibliotecas, librerías u otras interfases, para crear el programa que va a ser ejecutado directamente por el procesador final.

2.1.4. Administración de la tabla de símbolos.

Un compilador al realizar cada una de las etapas del proceso de compilación registra los identificadores utilizados en el programa fuente y reúne información sobre los distintos atributos de cada identificador. Estos atributos pueden proporcionar información sobre la memoria asignada a dicho identificador, su tipo, su ámbito (parte del programa donde tiene validez) y en caso de nombres de procedimientos, características como el número y tipo de argumentos, el método de pasar cada argumento y el tipo que devuelve, si lo hay.

Se le llama tabla de símbolos a la estructura de datos que contiene un registro por cada identificador con los campos necesarios para almacenar los atributos de cualquiera de los identificadores.

Cuando el analizador léxico detecta un identificador en el programa fuente este identificador se introduce en la tabla de símbolos y las diferentes fases actualizan la información sobre los identificadores en la tabla.

2.1.5. Detección e información de errores

Cada fase puede encontrar errores. Sin embargo después de detectar un error cada fase debe tratar de alguna forma ese error para poder continuar con la compilación, permitiendo así la detección de más errores en el programa fuente.

En la sección 4.4 se describe como se manejaron los errores en este compilador.

2.2. Análisis Léxico

2.2.1. Función del analizador léxico

Esta fase reconoce, verifica y clasifica, dentro del lenguaje específico, los diversos componentes de un programa fuente; desarrollando un mecanismo para dicho objetivo. Su función principal consiste en leer el texto fuente y elaborar como salida componentes léxicos (tokens) que son utilizados por el analizador sintáctico.

Las funciones de léxico son: Reconocer identificadores y palabras clave; agrupar los componentes de un número; eliminar espacios en blanco, de nueva línea o comentarios; así como generar una lista de los diferentes componentes, incluyendo para cada uno de ellos sus atributos generales. Además de las funciones anteriores, la fase de léxico también relaciona los mensajes de error del compilador con el programa fuente.

Para construir un lenguaje de programación es necesario especificar como se construyen sus expresiones y los componentes de dichas expresiones.

2.2.2. Reconocimiento de componentes léxicos

El objetivo de la fase de léxico es construir un analizador que aisle el lexema para el siguiente componente léxico de entrada y que produzca como salida el token apropiado y el valor de sus atributos (los que se conozcan hasta ese momento).

Para especificar como se construye un componente léxico se produce un diagrama de flujo estilizado, llamado diagrama de transición; los diagramas de transición están basados en la teoría de autómatas. Varios tokens pueden ser especificados a través del mismo diagrama pero el resultado no debe variar.

Las posiciones en un diagrama de transición se representan con un círculo y se llaman estados. Los estados se conectan mediante flechas, llamadas aristas. Las aristas que salen de los estados tienen etiquetas que indican los caracteres de entrada que pueden aparecer después de haber llegado, en el diagrama de transición, a tales estados. La etiqueta otro se refiere a cualquier carácter que no ha sido indicado por alguna arista de las que salen del estado analizado. Los diagramas de transición en esta etapa son deterministas; es decir, ningún símbolo puede coincidir con las etiquetas de dos o más aristas que salen de un mismo estado.

Un estado que se etiqueta como de inicio, o estado cero, es aquel donde reside el control cuando se empieza a reconocer un componente léxico. Algunos estados pueden tener acciones asociadas, estas se ejecutan cuando el flujo de control alcanza dicho estado. Al llegar a un estado se lee el siguiente carácter de entrada; si existe alguna arista, que sale del estado en curso, cuya etiqueta concuerde con el carácter de entrada, se va al estado apuntado por dicha arista; de otro modo se genera un error, el número de error se indica, para este trabajo, dentro de un rectángulo sencillo.

Un componente léxico ha sido identificado cuando se llega a un estado llamado aceptor, los estados aceptores fueron representados con rectángulos dobles en los diagramas de transición.

En 4.1 se describe un ejemplo de un diagrama de transición.

2.2.3. Implantación de diagramas de transición

Los métodos de desarrollo para la fase de léxico pueden ser: programación pura o a través del uso de una matriz de transición.

En el método de programación pura a cada estado le corresponde un segmento de código, si hay aristas que salen de un estado, entonces su código lee un carácter y selecciona una arista para continuar. Las aristas de los diagramas de transiciones se encuentran seleccionando repetidamente el fragmento de código para un estado y ejecutando ese fragmento de código para determinar el siguiente estado.

La secuencia de diagramas de transición se convierten en segmentos de código en donde cada segmento es asociado a un diagrama y su código es proporcional al número de estados y aristas del diagrama. Como los diagramas de transición se construyen a partir del estado cero, este aparece como un punto origen del programa y a partir de ahí se procesan repetidamente los segmentos de código necesarios hasta que aparece un estado aceptor para un token válido o un error.

El método de programación usando una tabla de transición consiste en crear una tabla en donde existe una fila por cada estado, una columna por cada símbolo de entrada y una última columna para los caracteres identificados como "otro". La manera de implementar este método se explica en la sección 4.1.

La representación en forma de tabla tiene varias ventajas: proporciona acceso rápido a las transiciones de un determinado estado, sí se desea agregar nuevos diagramas de

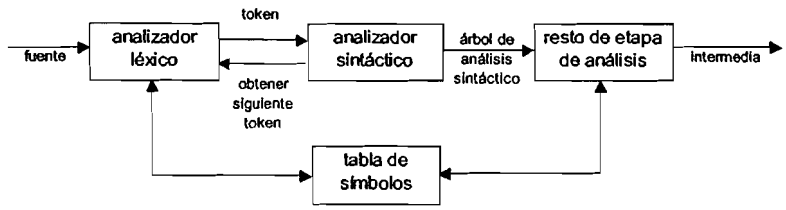


Figura 2.2: Posición del analizador sintáctico en el modelo de compilación

transición o modificar los existentes; el tiempo de depuración de la tabla puede disminuir considerablemente cuando se desea manejar un gran número de tokens; si la información necesita ser analizada por diferentes personas o en diferentes desarrollos, está queda establecida de manera clara . El inconveniente es que, la tabla en sí, puede ocupar gran cantidad de espacio cuando el alfabeto de entrada es muy grande.

2.3. Análisis Sintáctico

2.3.1. Introducción

Cada lenguaje de programación tiene reglas que preescriben la estructura sintáctica de programas bien formados. El análisis sintáctico es el proceso a través del cual las secuencias de tokens son examinadas para determinar si dichas secuencias obedecen las convenciones de estructura explícitas en la definición sintáctica del lenguaje.

Desde un conjunto de reglas sintácticas es posible diseñar analizadores de construcción automáticos definidos por dichas reglas de sintaxis. En el modelo de compilación, el análisis sintáctico obtiene un conjunto de tokens del analizador léxico como se muestra en la figura 2.2, verifica que ese conjunto pueda ser generado por la gramática del programa fuente y proporciona como salida un árbol que representa la estructura heredada en el programa fuente.

Aho, Ullman y Sethi en [Aho, Sethi y Ullman, 1972] describen que existen tres clases de lenguajes: Los libres de contexto, los regulares y los deterministas independientes de contexto. La mayoría de las sintaxis de lenguajes de programación modernos pueden ser descritas por gramáticas independientes de contexto.

Hay dos métodos principales para definir lenguajes: el generador y el reconocedor. [Aho, Sethi y Ullman, 1972]

2.3.2. Generadores

Aho, Ullman y Sethi en [Aho, Sethi y Ullman, 1972] definen un lenguaje L como un conjunto de cadenas de longitud finita sobre un alfabeto finito Σ.

Las gramáticas son la clase más importante de generadores. Una gramática es un sistema matemático para definir un lenguaje, así como un mecanismo para dar una estructura útil a las oraciones en el lenguaje. Una gramática para un lenguaje L usa dos conjuntos de símbolos finitos: El conjunto de los símbolos no terminales, frecuentemente denotado por N, y el conjunto de los símbolos terminales Σ. El conjunto de los símbolos terminales es el alfabeto sobre el que un lenguaje es definido. Los símbolos no terminales son usados en la generación de palabras.

El corazón de una gramática es un conjunto P de reglas o producciones que describen como se generan las oraciones del lenguaje. Una producción es un par de cadenas, o formalmente, un elemento de $(N \bigcup \sum)^* N (N \bigcup \sum)^* * (N \bigcup \sum)^*$. Que significa, que el primer componente es cualquier cadena que contiene al menos un no-terminal y el segundo componente es cualquier cadena.

En [Aho, Sethi y Ullman, 1972] Aho, Ullman y Sethi escriben la siguiente definición:

Una *gramática* es un tupla de 4 $G = (N, \sigma, P, S)$ donde:

1. N es un conjunto de *símbolos no terminales.*
2. Σ es un conjunto finito de *símbolos terminales*, sin intersección con N
3. P es un subconjunto de
   $(N \bigcup \sum)^* N (N \bigcup \sum)^* * (N \bigcup \sum)^*$
   Un elemento (α, β) en P se escribirá $\alpha \rightarrow \beta$ llamado $producc in$
4. S es un símbolo en N llamado símbolo de inicio.

Se sigue la regla convencional de especificar las gramáticas dando una lista de sus producciones, donde las producciones del símbolo inicial se listan primero. Un nombre en *cursiva* es un no terminal, y se supone que cualquier nombre o símbolo que no este en cursiva es un componente léxico.

Se dice que una producción es *para* un no terminal si el no terminal aparece en el lado izquierdo de la producción. Una cadena de componentes léxicos es una cadena de cero

o más componentes léxicos. La cadena que contiene cero componentes léxicos, que se escribe como ϵ, recibe el nombre de cadena *vacía*.

De una gramática se derivan cadenas empezando con un símbolo inicial y reemplazando repetidamente un no terminal por el lado derecho de la producción para ese no terminal. Las cadenas de componentes léxicos derivadas del símbolo inicial forman el *lenguaje* que define la gramática.

Además de servir para especificar la sintaxis una gramática independiente de contexto se puede usar para guiar la traducción de programas.

La clase más común de generador es la gramática Chomsky.

El análisis sintáctico es el proceso de determinar si una cadena de componentes léxicos puede ser generada por una gramática.

Un árbol sintáctico indica gráficamente como del símbolo inicial de una gramática deriva una cadena de lenguaje. Si el no terminal A tiene una producción $A \rightarrow XYZ$, entonces un árbol de análisis sintáctico puede tener un nodo interior etiquetado con A tres hijos etiquetados con X, Y, Z, de izquierda a derecha.

2.3.3. Reconocedores

Un segundo método de proveer una especificación finita para un lenguaje es definir un reconocedor para el lenguaje. En esencia un reconocedor es simplemente un procedimiento altamente estilizado para definir un conjunto. Se puede dibujar como en la figura 2.3.

Un reconocedor contiene tres partes: una cinta de entrada, un estado finito de control y una memoria auxiliar.

Se puede considerar que, la cinta de entrada, puede ser dividida en una secuencia de cintas cuadradas; cada cinta cuadrada contiene un símbolo de entrada de un alfabeto de entrada finito. Ambos, la cintas cuadradas de más a la izquierda y de más a la derecha pueden estar ocupados por marcadores únicos, o podría haber un marcador de final a la derecha y ninguno a la izquierda, o podría no haber marcadores de final.

Hay una cabeza de lectura, en la que se puede leer una entrada cuadrada en un instante de tiempo. En un movimiento del reconocedor, la cabeza de entrada puede mover un cuadro a la izquierda o a la derecha y dejarlo fijo. Un reconocedor que nunca puede mover su cabeza de entrada a la izquierda es llamado reconocedor en un sentido.

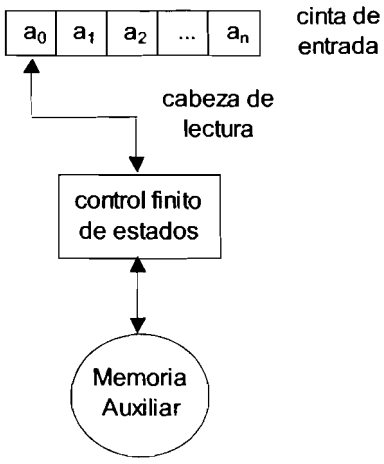


Figura 2.3: Un Reconocedor.

Normalmente, se asume que, la cinta de entrada es de solo lectura, lo que significa que una vez que la cinta de entrada es cargada sus símbolos no se pueden cambiar. Sin embargo es posible definir reconocedores que tengan una cinta de lectura y escritura.

La memoria de un reconocedor puede ser cualquier tipo de fuente de datos. Se asume que hay una alfabeto de memoria finito y que la memoria contiene solo símbolos del alfabeto de memoria finito en alguna organización de los datos. También se asume que en un instante de tiempo finito, se puede describir el contenido y la estructura de la memoria, aunque con el paso del tiempo la memoria puede llegar a ser arbitrariamente mayor.

El comportamiento de la memoria auxiliar para algunos reconocedores se caracteriza por dos funciones: una función que almacena y otra función que trae. Se asume que el trabajo de la *función que trae* es hacer un mapeo de la memoria desde el conjunto de configuraciones posible a un conjunto finito de *símbolos de información*, los cuales pueden ser como el alfabeto de la memoria. La función de almacenar hace un mapeo que describe como se va a alterar la memoria.

El tipo de memoria determina el nombre del reconocedor.

El corazón de un reconocedor es el estado de control finito el que se puede entender como un programa que dicta el comportamiento de un reconocedor. El control se puede representar como un conjunto de estados junto con un mapeo que describen como los estados cambian de acuerdo con el actual símbolo de entrada (es decir, el único bajo la cabeza de entrada) y la información cargada desde la memoria. El control también determina en

que dirección la cabeza de entrada va a cambiar y que información se va a almacenar en la memoria.

Un reconocedor opera haciendo una secuencia de movimientos. Al inicio de un movimiento, se lee el símbolo de entrada actual y se revisa la memoria para encontrar el significado de la función de carga. El símbolo de estrada actual y la función cargada desde la memoria junto con el estado actual del control determinan que movimiento se va a realizar. El movimiento consiste de:

1. Mover la cabeza de entrada un cuadro a la derecha, uno a la izquierda o mantener la cabeza de entrada en una posición fija
2. Almacenar la información dentro de la memoria, y
3. Cambiar el estado del control

El comportamiento de un reconocedor se puede describir de manera adecuada en términos de las configuraciones del reconocedor. Una configuración es una fotografía del reconocedor describiendo

1. El estado de control finito
2. El contenido de la cinta de entrada juntamente con la ubicación de la cabeza de entrada y
3. El contenido de la memoria.

El control finito de un reconocedor puede ser determinista o no determinista. Si el control es *no determinista*, en cada configuración hay un conjunto finito de posibles movimientos que el reconocedor puede hacer.

El control es *determinista* si en cada configuración hay a lo sumo un movimiento posible. Los reconocedores no deterministas son una abstracción matemática muy útil, pero, desafortunadamente frecuentemente difíciles de simular en la práctica.

En la *configuración inicial* de un reconocedor se especifica el estado inicial del control finito, la cabeza de entrada está leyendo el símbolo de más a la izquierda de la cinta de entrada y ha sido especificado el contenido inicial para la memoria.

En la *configuración final* el control finito está en uno de un conjunto de estados finales y la cabeza de entrada en el símbolo de más a la derecha de la cinta de entrada. Frecuentemente la memoria debe satisfacer ciertas condiciones sí la configuración será considerada como configuración final.

Se dice que el reconocedor acepta *una cadena de entrada* w sí, comenzando desde la configuración inicial con w en la cinta de entrada, el reconocedor puede hacer una secuencia de movimientos y terminar en una configuración final.

El lenguaje definido por un reconocedor es el conjunto de las cadenas de entrada que acepta. Para cada clase de gramática en la jerarquía de Chomsky hay una clase natural de reconocedores que define la misma clase de lenguajes. Esto reconocedores son los autómatas finitos, autómata de pila (pushdown automata), autómata lineal acotado (linear bounded automata) y las máquinas de Turing. La clasificación de Chomsky es:

1. Un lenguaje L es lineal a la derecha si y solo sí L es definido por un autómata finito (determinista de un camino).

2. Un lenguaje L es independiente de contexto si y solo sí L es definido por un autómata de pila (pushdown) (no determinista de un camino).

3. Un lenguaje L es sensible al contexto si y solo sí L es definido por un autómata acotado linealmente (no determinista de dos caminos).

4. Un lenguaje L es enumerable de dos caminos si y solo sí L es definido por una máquina de Turing.

2.4. Generador de código

La primer etapa del proceso de síntesis es la fase de generación de código.

Esta fase toma como entrada una representación intermedia del programa fuente y produce como salida un programa objeto equivalente. La estructura de datos o árbol construido por el analizador de sintaxis se usa en la fase de síntesis para generar una traducción del programa de entrada.

Aho en [Aho, Sethi y Ullman, 1986] escribe que las exigencias tradicionalmente impuestas a un compilador son duras. El código de salida debe ser correcto y de gran cali-

dad, lo que significa que debe utilizar de forma eficaz los recursos de la máquina objeto. Además el generador de código mismo debe ejecutarse eficientemente.

El problema de generar código óptimo es muy difícil de modelar matemáticamente. En la práctica, hay que conformarse con reglas heurísticas que generen código bueno. La elección de las reglas heurísticas es importante, ya que un algoritmo de generación de código cuidadosamente diseñado puede producir fácilmente código que sea varias veces más rápido que el producido por un algoritmo diseñado precipitadamente.

La naturaleza del conjunto de instrucciones de la máquina objeto determina la dificultad de la selección de instrucciones. Es importante que el conjunto de instrucciones sea uniforme y completo; si la máquina objeto no apoya cada tipo de datos de una manera uniforme, entonces cada excepción a la regla exige un tratamiento especial.

Las instrucciones que implican operandos con registros son generalmente más cortas y rápidas que las utilizan operandos en memoria, entonces, utilizar eficientemente los registros es fundamental para generar un buen código.

Aho en [Aho, Sethi y Ullman, 1986] comenta que el uso de registros se divide a menudo en dos subproblemas:

1. Durante la *asignación de los registros*, se selecciona el conjunto de variables que residirá en los registros en un momento del programa.
2. Durante una fase posterior de *asignación a los registros*, se escoge el registro específico en el que residirá una variable.

La familiaridad con la máquina objeto y su conjunto de instrucciones es un pre-requisito para diseñar un buen generador de código.

2.5. Optimizador de código

La función del optimizador de código es lograr que el código directamente producido por los algoritmos de compilación se ejecute más rápidamente o que ocupe menos espacio.

Wolfe en [Wolfe, 1996] comenta que en la realidad, en la mayoría de los lenguajes, este objetivo solo se alcanza en pocos casos y difícilmente, además, escribe que, la optimización de código no debe cambiar el comportamiento de un programa; pero que hay que tener cuidado al definir el comportamiento que se desea conservar.

Hay varias formas en que un compilador puede mejorar un programa sin modificar la función que calcula. La eliminación de subexpresiones comunes, la eliminación de código inactivo y el cálculo previo de constantes son ejemplos comunes de dichas transformaciones que preservan la función:

1. Una expresión E se denomina *subexpresión común* si E ha sido previamente calculada y los valores de las variables dentro de E no han cambiado desde el cálculo anterior. Se puede evitar recalcular la expresión si se puede utilizar el valor calculado previamente.

2. Una variable está activa si su valor puede ser utilizado posteriormente; en caso contrario está inactiva en ese punto. Una idea afín es el código inactivo o inútil, proposiciones que calculan valores que nunca llegan a utilizarse; aunque es improbable que el programador introduzca código inactivo intencionalmente, puede aparecer como resultado de transformaciones anteriores.

Capítulo 3

Programación en Código G

3.1. Planeación de un programa

Smid en [Smid, 2003] describe algunos pasos básicos que deben ser considerados para planear un programa: Información inicial (características de la máquina-herramienta), complejidad de la parte, programación por computadora ó manual, procedimiento típico de programación, dibujo de la parte ó información de ingeniería, hoja de métodos ó especificación de materiales, secuencia de maquinado, selección de la herramienta, ajuste de la parte, decisiones tecnológicas, diagrama de trabajo y cálculo y consideraciones de calidad. También sugiere un procedimiento de programación compuesto por los siguientes puntos: Estudio de la información inicial (dibujos y métodos), evaluación del material para fabricar la pieza y del que hay que remover, especificaciones de la máquina-herramienta, dispositivos del sistema de control, secuencia de operaciones de maquinado, selección y arreglo de las herramientas de corte, configuración de la parte, datos tecnológicos (alimentación, velocidad, etc.), determinar el camino de la herramienta, dibujos de trabajo y cálculos matemáticos, escribir el programa y prepararlo para transferirlo al CNC, realizar pruebas y depuración del programa y documentarlo.

3.2. Estructura de un programa CNC

Un programa CNC está compuesto de una serie de instrucciones secuenciales relacionadas entre sí para el maquinado de una pieza. Cada instrucción tiene que ser conforme

a las especificaciones de la máquina-herramienta.

Hay cuatro términos básicos usados en programación CNC:

Caracter ⇒ *Palabra* ⇒ *Bloque* ⇒ *Programa*

Un carácter puede ser un dígito, una letra o un símbolo usado para programar. Una palabra es un conjunto de caracteres que agrupados tienen un significado especial; algunas palabras son la posición de un eje, la especificación de la velocidad de alimentación, alguna función preparatoria o alguna función miscelánea. Un bloque es una instrucción particular para el sistema CNC y está compuesto de una o varias palabras. En programación en código G un bloque es una línea de entrada, pero en la salida generada por el sistema es distinto, como se describirá en 4.5.

En este sistema los datos de entrada en un bloque pueden ser especificados en cualquier orden como lo comenta Smid en [Smid, 2003], aunque es una práctica ubicar las palabras en un orden lógico dentro de un bloque. Una estructura común de un bloque en un programa contiene las palabras en el siguiente orden. No es necesario especificar todos los datos cada vez, solo cuando requieran.

Descripción	Dirección
Número de bloque	N
Funciones preparatorias	G
Funciones auxiliares	M
Funciones de movimiento de ejes	X Y Z A B C U V W...
Palabras relativas a los ejes	I J K R Q...
Velocidad, alimentación o función de herramienta	S F T

3.3. Números de secuencia

Los bloques en un programa CNC se pueden referenciar con un número para una orientación más fácil dentro del programa. La dirección de programa para un número de bloque es la letra N, seguida hasta de 7 dígitos de 1 a 9999999; en este sistema un número de bloque siempre debe ser mayor al anterior número de bloque especificado. También se utiliza para dar a conocer al operador a través de la interface en pantalla que número de

operación se está llevando realizando en la máquina-herramienta.

3.4. Identificación de un programa

El primer bloque utilizado en cualquier programa es comúnmente un número de programa; aunque este no siempre es necesario. La dirección utilizada en este sistema para almacenar el número de programa es la letra O y el rango para el número es entre uno y cien mil.

3.5. Comentarios de programa

Se pueden ubicar comentarios dentro de un programa para CNC; la manera de indicar que un texto es comentario, es decir que no se va a utilizar por el CNC solo es información auxiliar, es encerrando el texto entre paréntesis cuando contiene una línea o más de una ó con un se indica que el resto de la línea escrita es comentario.

3.6. Funciones Preparatorias

La dirección de programa G identifica una función preparatoria, su objetivo es activar o preparar el sistema de control para cierta condición, modo o estado de operación. Los códigos G pueden ser modales o no modales; que un comando sea modal significa que este se va a mantener en cierto modo hasta que sea cancelado por otro modo; en la tabla 3 se muestran las funciones preparatorias implementadas en este sistema, las que tienen un número diferente de cero en el campo Gr(grupo) son modales. Un código G en un grupo modal reemplaza cualquier código G del mismo grupo.

Varias funciones preparatorias pueden ser usadas en un mismo bloque verificando que no haya conflicto lógico entre ellas. El propósito de las funciones preparatorias es seleccionar un modo entre dos o más de operación. Si, por ejemplo, la función G00 se selecciona, la función que se refiere al movimiento rápido está activa, por lo que es imposible tener un movimiento rápido y uno de corte al mismo tiempo, no se puede tener G00 y G01 activas al mismo tiempo.

Las funciones preparatorias en el grupo 00 no son modales. Son activas solo en el bloque en el que aparecen. Si se requiere una función G no modal en varios bloques consecutivos, tiene que ser programada en cada uno de esos bloques; en la mayoría de los comandos no modales esto no ocurre con frecuencia.

3.6.1. Entrada de dimensiones

Las direcciones en un programa CNC que se refieren a la posición de la herramienta en un momento dado son llamadas *palabras de coordenada*. Las palabras de coordenadas siempre toman un valor dimensional usando las unidades activas, métricas o inglés.

Las dimensiones en un programa asumen dos atributos:

1. Unidades de dimensión, sistema inglés o métrico.
2. Referencia dimensional, absoluta o incremental.

Las dimensiones de dibujo pueden ser usadas en un programa en cualquiera unidades métricas o Inglesas. Para seleccionar una entrada específica de dimensionamiento se requiere una función preparatoria:

G20	Selecciona unidades Inglesas (pulgadas o pies)
G21	Selecciona unidades métricas (milímetros y metros)

Una dimensión en cualquier unidades de entrada tiene que tener un punto de referencia específico. Lay dos tipos de referencias en programación:

1. La referencia a un punto común en la parte. Conocido como el origen para entrada absoluta.
2. La referencia a un punto previo en la parte. Conocido como la última posición de la herramienta para entrada incremental.

Hay dos funciones preparatorias disponibles para los valores de dimensionamiento:

G90	Modo de dimensionamiento absoluto
G91	Modo de dimensionamiento incremental

Para los controladores de los tornos FANUC, la representación del modo absoluto en los ejes son X y Z, sin la función G90; para como modo incremental son los ejes U y W, sin la función G91.

3.6.2. Posicionamiento Rápido

Los movimientos de posicionamiento son necesarios pero no-productivos, no pueden ser totalmente eliminados y tienen que ser manejados lo más eficientemente posible. Su principal objetivo es acortar el tiempo de posicionamiento entre las operaciones que no son de corte, cuando la herramienta de corte no está en contacto con la parte. Las operaciones de movimiento rápido generalmente abarcan cuatro tipos de posicionamiento:

1. Desde la posición de l cambio de herramienta a la parte
2. Desde la parte hasta la posición del cambio de herramienta
3. Movimientos para brincar obstáculos
4. Movimientos entre las diferentes posiciones de la parte

Al movimiento rápido también algunas veces se le llama movimiento de posicionamiento , es el método para mover la herramienta de corte de una posición a otra a una velocidad rápida de la máquina. La velocidad de movimiento puede ser la misma para cada eje o diferente. Un movimiento rápido se puede ejecutar como un movimiento de un solo eje o como uno de dos o más ejes de manera simultánea. Se puede programar en modo absoluto o incremental de dimensionamiento y el husillo puede o no estar en movimiento. Se usa la función preparatoria G00 para inicializar el modo de posicionamiento rápido. No se necesita la función de alimentación para ejecutar una función G00.

3.6.3. Regreso al cero de la máquina

Programadores y operadores entienden el término *posición de referencia de la máquina* como sinónimo de *home* o *cero de la máquina*. Esta es la posición de todos los lados de

la máquina en el extremo para cada eje. Al menos un eje debe ser especificado con ambas funciones que especifica un punto intermedio para el movimiento.

Se implementaron dos funciones preparatorias relacionadas con el zero de la máquina:

G28	Regreso al punto de referencia del cero de la máquina
G29	Regreso desde el punto de referencia del cero de la máquina

La función G28 mueve el eje o ejes especificados a la posición de *home* a la velocidad de movimiento rápido, lo que significa que se asume una función G00. Los valores de los ejes asociados a la función indican un punto intermedio por el que ha de pasar el movimiento.

La función preparatoria G29 es exactamente lo opuesto a la función G28, esta regresa la herramienta a su posición original vía un punto intermedio.

3.6.4. Interpolación Circular

EL formato de programación para una interpolación circular del camino de herramienta tiene que incluir varios parámetros, sin los cuales la tarea de corte sería imposible. Los parámetros importantes se definen como:

1. La dirección de corte del arco (CW o CCW)
2. El inicio y final del arco
3. El centro del arco y el valor del radio

También hay que especificar la velocidad de alimentación. El movimiento desde el eje del plano vertical hacia el eje del plano horizontal es en sentido horario, el movimiento al contrario es el antihorario. Hay dos funciones preparatorias asociadas con programar un arco:

G02	Movimiento circular en sentido horario
G03	Movimiento circular en sentido antihorario

Cuando se activa una función G02 o G03 por un programa de CNC, el movimiento de herramienta actual activo es cancelado automáticamente. Todos los movimientos de camino de herramienta tienen que ser programados con una velocidad de alimentación de corte.

Un radio único se alcanza programando la dirección R para una entrada directa de radio, o usando los vectores de centro de arco IJK. El vector I es la distancia especificando dirección, medida desde el punto de inicio del arco, paralela al eje X. El vector J es la distancia especificando dirección, medida desde el punto de inicio del arco, paralela al eje Y. El vector K es la distancia especificando dirección, medida desde el punto de inicio del arco, paralela al eje Z. La distancia entre el punto de inicio del arco y el centro se mide como una distancia incremental entre los dos puntos. También se puede usar la dirección R en un programa, con un signo negativo para indicar cualquier arco mayor a 180°.

3.6.5. Función del husillo

La función de programa que se refiere a la velocidad del husillo es controlada en el sistema CNC por la dirección S. El rango de valores que puede tomar en algunos controladores es de 1 a 9999 y en otros de 1 a 99999, en este sistema se puede modificar dicho rango como el controlador lo permita. El rango de la velocidad máxima disponible en el control tiene que ser mayor que el rango de velocidad máxima en el husillo.

Si la velocidad del husillo y su rotación son programadas en el mismo bloque, comienzan simultáneamente. Si se programan en diferentes bloques, el husillo no comenzará su rotación hasta que ambas funciones se hayan procesado.

3.6.6. Control de la velocidad de alimentación

La velocidad de alimentación de corte es la velocidad a la cual la herramienta de corte remueve el material por la acción de corte. Hay dos tipos de velocidades de alimentación usadas en programación de CNC:

1. velocidad de alimentación por minuto
2. velocidad de alimentación por revolución

En este sistema las funciones para programar los diferentes modos de esta función son:

G94	Velocidad de alimentación por minuto
G95	Velocidad de alimentación por revolución
G98	Velocidad de alimentación por minuto
G99	Velocidad de alimentación por revolución

3.6.7. Funciones de registro

La forma tradicional para indicarle al sistema de control donde está localizada cada herramienta dentro del area de trabajo de la máquina, antes de que esta pueda ser usada. Este método requiere una función llamada registro de posición.

Las funciones preparatorias para el registro de posición de la herramienta en este sistema son:

G50	Función de registro de posición (usado en fresa)
G92	Función de registro de posición (usado en torno)

En programación moderna de CNC ambos registros fueron reemplazados por características mucho más sofisticadas y flexibles, los Offsets de Trabajo (G54 a G59).

En un bloque que contiene una función G50 o G92 no ocurre ningún movimiento de máquina.

El formato de la función G50 y G92 es:

```
G50   X.. Z..
G92   X.. Z..
```

En este sistema se permite usar cualquiera de las dos funciones mencionadas para definir el registro de posición.

3.7. Funciones Misceláneas

3.7.1. Descripción y propósito

La dirección de programa M identifica una función miscelánea, algunas veces se le llama función de la máquina. Varias operaciones físicas de la máquina CNC tienen que

ser controladas por el programa para asegurar el maquinado automático completo. Esta funciones generalmente usan la dirección M e incluyen las siguientes operaciones:

Función M	Operación
Rotación del husillo	CW o CCW
Cambio de rango de engrane	Bajo/Medio/Alto
Cambio automático de herramienta	ATC
Cambio automático de rack	APC
Operación del refrigerante	encendido/apagado
Movimiento del cabezal móvil	retrasado o adelantado

Estas operaciones varían entre máquinas, debido a los diferentes diseños por diversos fabricantes. Todas las máquinas-herramienta diseñadas para remover metal por corte tienen ciertas características comunes y capacidades como: la rotación del husillo puede tener tres -y solo tres- elecciones posibles dentro de un programa (rotación del husillo en sentido horario, rotación del husillo en sentido antihorario y paro de husillo) ó el refrigerante puede estar encendido o apagado.

El uso de las funciones misceláneas cae principalmente en dos grupos: Control de las funciones de la máquina y control de la ejecución del programa.

En la tabla 3 se muestran las funciones misceláneas implementadas en este sistema.

Varias funciones misceláneas pueden ser activadas dentro de un mismo bloque mientras ellas no pertenezcan al mismo grupo en el campo Gr de la tabla y en este sistema todas ellas pueden ser programadas como la única función en un bloque. Para conocer la duración de una función M es útil conocer cuanto tiempo estará activa; algunas funciones solo están activas en el bloque en el que aparecen y otras tienen efecto hasta que son canceladas por otra función miscelánea.

En 3.7.2 y 3.7.3 se describen las funciones misceláneas más importantes.

3.7.2. Funciones de Programa

Las funciones misceláneas que controlan el procesamiento del programa pueden ser usadas para interrumpir el proceso temporalmente (en la mitad de la ejecución) o permanentemente (al final de la ejecución del programa).

1. **Paro de programa** La función M00 está definida como un paro incondicional. En cualquier momento que el sistema de control encuentre esta función durante el procesamiento del programa, la máquina-herramienta detendrá las operaciones automáticas: el movimiento de los ejes, la rotación del husillo y las funciones del refrigerante. El control no se inicializa cuando se procesa una función M00. Los datos activos del programa se retienen (alimentación, posición de los ejes, velocidad del husillo, etc.). La función M00 cancela la rotación del husillo y la función del refrigerante.

2. **Paro condicional de programa** La función miscelánea M01 es un paro de programa opcional o condicional. Es semejante a M00 con una diferencia, cuando se encuentra una función M01 en el programa, el procesamiento de este no se detiene, a menos que el operador interfiera vía panel de control. El interruptor de palanca o de botón en el panel puede estar en posición de encendido o apagado; cuando está encendido el proceso se detendrá, cuando está apagado el proceso no se detendrá.

3. **Fin de programa** Cada programa tiene que incluir una función para su finalización; hay dos funciones con para este objetivo: M30 y M02, estas funcionan de manera similar. La función M02 termina el programa pero no causa un regreso al inicio de este. La función M30 también termina el programa pero regresa a su inicio. Cuando un control lee una función M02 o M30 en un programa, cancela todos los movimientos de los ejes, la rotación del husillo, las funciones de refrigerante y normalmente inicializa el sistema a las condiciones por omisión.

3.7.3. Funciones de Máquina

1. **Funciones de refrigerante** La mayoría de las operaciones para remover metal requieren que la herramienta de corte sea rociada con un refrigerante apropiado. Para controlar el flujo del refrigerante en el programa hay tres funciones misceláneas:

M07	Mezcla conectada
M08	Flujo conectado
M09	Mezcla o flujo encendido

Mandar un flujo sobre el filo de corte de la herramienta es importante para disipar el calor, remover la rebaba y para lubricación.

2. **Funciones del husillo** La mayoría de los husillos pueden rotar en ambas direcciones, sentido horario (CW) y antihorario (CCW). Esta dirección siempre es relativa a un

punto de vista estándar; desde el lado del husillo como la dirección a través de la linea central de este, hacia su cara. La rotación CW se programa como M03 y CCW como M04.

3. **Selección de rango de engranes** La distribución de las funciones misceláneas depende del rango de engranes disponibles en el torno.

Rango	Función M	Rango de engranes
1 disponible	N/A	Ninguno programado
2 disponibles	M41	Rango bajo
	M42	Rango alto
3 disponibles	M41	Rango bajo
	M42	Rango medio
	M43	Rango alto
4 disponibles	M41	Rango bajo
	M42	Rango 1 medio
	M43	Rango 2 medio
	M44	Rango alto

Capítulo 4

Desarrollo de análisis y síntesis del compilador

4.1. Desarrollo del Análisis Léxico

Sabemos de 2.2.1 que el objetivo de esta etapa es obtener cada conjunto de caracteres que para nuestro lenguaje tiene algún significado: un token; así como asociar a cada token la información reconocida en dicha identificación. Para resolver esto se utilizó el método de tabla de transición. Además conocemos de 2.2.2 que la manera de establecer los formatos válidos para los identificadores aceptados por el lenguaje fuente es a través de diagramas de transición. Para este sistema los diagramas se pueden consultar en 5.5.

Cada uno de los diagramas parten del estado 0 o estado inicial. Tomando como ejemplo el diagrama de transición utilizado para reconocer las funciones preparatorias en 13 se puede observar que, a partir del estado cero, sí el primer caracter identificado es una ''G'', se avanza al estado de transición 1, sí después se identifica cualquier otro caracter diferente de un dígito, se va al error 801; si enseguida del caracter ''G'' el siguiente caracter es un dígito se pasa al estado 2 y a partir de ahí, permaneciendo en el estado 2, se agrupan cada uno de los dígitos consecutivos que aparezcan hasta que se identifique cualquier otro caracter diferente de un dígito, cuando se lee este caracter, se devuelve el token formado por el caracter ''G'' y todos los dígitos que le siguieron además de un número 107 que indica el número de token identificado. Siguiendo la misma lógica se pueden analizar cada uno de los diagramas de transición para establecer así, un número de token diferente para cada conjunto de caracteres con significado particular.

Analizando los diagramas de transición podemos ver que todas las funciones preparatorias son identificadas a través del número de token 103, así como las funciones misceláneas con el número de token 107. También leímos en 2.2.1 que esta etapa también verifica que las palabras reservadas formen parte del lenguaje. Una vez que se ha identificado un token 103 se busca a través de esta etapa en la tabla 3 para conocer el número de palabra reservada asociado a este token. De la misma manera en 4 para las Funciones Misceláneas reconocidas a través del número 107. Con excepción de las funciones preparatorias y misceláneas, los demás números de token definidos como válidos en 5.5 ya no siguen otro proceso de identificación en la etapa de léxico. Para las palabras identificadas como funciones preparatorias o misceláneas la información asociada al token, encontrada en las tablas en el proceso de identificación de palabra reservada, también acompaña al token para su uso en las siguientes etapas del proceso de compilación.

4.1.1. Implantación en tabla de diagramas de transición

Para vaciar la información de los diagramas a la tabla, se toma en orden el primer diagrama que contienen el estado cero y la primer arista que sale de dicho estado, para cada uno de los caracteres asociados a ella, se escribe, en el número de columna que le corresponda a cada uno de esos caracteres, el número del estado, el número de estado aceptor o el número de error al que apunta la primer arista; enseguida se toma la segunda arista que sale del estado inicial y se escribe nuevamente, en el número de columna asociado a cada caracter, el número del estado, el número de estado aceptor o el número de error al que apunta esa segunda arista. El proceso se repite hasta terminar con todas las aristas que salen del estado inicial. Así se vacía la información a la tabla para cada uno de los estados y cada una de las arista que salen de ellos, tomando como número de fila al número de estado del cual se está realizando el vaciado.

4.1.2. Exploración de la tabla de transición

Ya en la etapa del análisis léxico de un programa, el proceso de exploración de la tabla se inicia en el renglón o estado cero. Al identificar el primer caracter se busca la columna definida para dicho caracter y se interpreta el contenido de la casilla como el renglón-estado hacia el cual se dirige el apuntador en la tabla de transición. Si el estado encontrado está definido como aceptor, se termina de obtener el token, estos estados fueron identificados en la tabla como mayores o iguales a 100 y menores a 200. Si el estado encontrado está definido como de transición, menor a 100, este indica el nuevo renglón-

estado para el apuntador en la tabla; se concatena el caracter encontrado; se realiza la lectura del siguiente caracter, quien define la columna donde se lee el siguiente estado-renglón; y se continua hasta encontrar un estado aceptor o un error; como se concatenan los carateres uno a uno, cuando se llega a un estado aceptor se conoce el token identificado así como su número correspondiente. Cualquier caracter que permite terminar de identificar un token válido es un delimitador para dicho token.

Al explorar la tabla de transición o al recorrer un diagrama de transición se puede encontrar un número que identifica un error, estos son los mayores o iguales a 800; si esto sucede, se guarda el número de error con su descripción en la lista de errores y se continua con la identificación del primer token en el siguiente bloque.

4.2. Análisis Sintáctico

4.2.1. Introducción

En 2.3.1 leímos que en el proceso de análisis sintáctico se verifica que las secuencias de tokens sigan un orden válido para el lenguaje y que cuando se ha establecido el conjunto de reglas sintácticas, se puede diseñar, un analizador de construcción automático para el lenguaje.

En esta etapa se utilizó un reconocedor como analizador de construcción, que, como se comenta en 2.3.3 se puede describir en términos de sus configuraciones: los datos de entrada, el control finito de estados o en este caso el proceso que se realiza con los datos de entrada y la memoria o condiciones prerrequisito para obtener la salida del reconocedor. También se uso una tabla de parseo para dejar establecido un procedimiento que facilite agregar posteriormente instrucciones con anidamientos.

Para las reglas del lenguaje en este proyecto la base fue el manual de programación CNC de Peter Smid citado en [Smid, 2003]; implementando, hasta el momento, las funciones que se consideraron más utilizadas en programación para torno.

Las funciones preparatorias implementadas son las que aparecen en la tabla 3 que por columna contiene la siguiente información:

1. Un número que corresponde al grupo de comandos al que pertenece, utilizado en esta fase de sintaxis;

2. el nombre de la función G;
3. el número de token, dato devuelto por la fase de léxico cuando le es solicitado un token del programa fuente;
4. la descripción en inglés y
5. la descripción en español de la función preparatoria

Las funciones misceláneas se muestran en la tabla 4 y su información comprende:

1. Un número que indica el grupo al que corresponde la función;
2. un número que puede ser 99, 1 o 0 dependiendo si dicha función debe ser activada al final del bloque, al inicio de el o durante el bloque;
3. el nombre de la función miscelánea;
4. el número de token;
5. la descripción de la función M en inglés y
6. la descripción de la función en español.

4.2.2. Desarrollo del Análisis Sintáctico

El desarrollo de esta etapa se llevo a cabo de la siguiente manera: La fase de sintaxis solicita a la fase de léxico uno a uno los tokens de un bloque al mismo tiempo que navega a través de la tabla de parseo que se muestra en 10; cuando se recibe un token comienza el recorrido de la tabla en el primer renglón y primera columna; mientras se realizan las acciones de navegación, estas, cambian el renglón y la columna de ubicación del analizador de acuerdo al token recibido, hasta encontrar un error ó un estado aceptor. Después se solicita otro token con el que se vuelve a navegar a través de la tabla de parseo y a realizar las acciones que el recorrido indique hasta encontrar un nuevo estado aceptor. Cuando se han verificado todos los tokens del bloque de entrada, el analizador sintáctico realiza aquellas revisiones que necesitan la información completa del bloque en memoria.

La tabla de parseo es un arreglo de tres dimensiones de (12,8,4). La profundidad uno contiene la acción para navegar a través de la tabla y utiliza la información de las profundidades dos y tres para sus operaciones. Si se cumple la condición uno, entonces, se realiza

la acción indicada en la profundidad cuatro, que permite: (a) actualizar los parámetros en memoria que no generan bloque alguno, pero que afectan la configuración de la memoria para procesos posteriores y (b) preparar la memoria para las verificaciones realizadas al final del bloque, pero que aún no se pueden realizar por que necesitan la información del bloque cargada de forma completa antes de terminar con la verificación de sintaxis. Si no se cumple la condición uno, se continúa la navegación a través de la matriz en el mismo renglón y la siguiente columna. Las acciones de navegación a través de la tabla de parseo se enumeran en 4.1. Para cambiar de un renglón a otro se utiliza la acción de navegación número cuatro.

Acciones para navegación a través de la Tabla de Parseo

Acción	Descripción
1	Sí lo que está en el renglón y columna actuales en la profundidad dos es el número de token en proceso, salta a la columna indicada en la profundidad tres.
4	Salta al renglón indicado en la profundidad dos, en la primer columna.
6	Genera el número de error indicado en la profundidad dos.
7	Proporciona un estado aceptor.

Las acciones de la cuarta columna en la tabla de parseo se describen en 4.2:

Acciones de sintaxis

Num Acción	Descripción
1	Activa el grupo de la función G (primer dato de la tabla 3) con el número de función.

continua en la siguiente página

Acciones de sintaxis

Num Acción	Descripción
2	Verifica en la memoria qué el parámetro indicado por el token en análisis no haya sido definido antes en el mismo bloque; si ya había sido definido, genera un error de duplicidad de parámetro; sino, define el parámetro. Si se trata de un parámetro de corte o posicionamiento evalúa el valor de acuerdo al sistema de unidades y los offsets activos. Verifica que el parámetro esté dentro de los límites válidos. Si es un parámetro N, verifica que el valor dado sea menor al último proporcionado. Si no hubo algún error con las revisiones anteriores dentro de esta acción, asigna a la correspondiente variable del registro en memoria el nuevo valor.
3	Asigna el valor de entrada dado por el token a la variable del registro en la memoria y si se trata de la función O (nombre del programa), valida que sea la primer función generada.
4	Verifica en la memoria qué el parámetro indicado por el token en análisis no haya sido definido antes en el mismo bloque; si ya había sido definido, genera un error de duplicidad de parámetro; sino, define el parámetro. Si se está definiendo el número de herramienta verifica que este número y su offset sean válidos. Si no hubo algún error durante las revisiones de esta acción asigna a la correspondiente variable del registro en memoria el nuevo valor.
5	Sí no ha sido activada antes en el bloque alguna otra función con parámetros, activa la función 4; si ya lo fue, genera el error 17.
6	Sí no ha sido activada antes en el bloque alguna otra función con parámetros, activa la función 28 ; si ya lo fue, genera el error 18.
7	Sí no ha sido activada antes en el bloque alguna otra función con parámetros, activa la función 29; si ya lo fue, genera el error 19.
8	Si no ha sido activada antes en el bloque alguna otra función con parámetros, activa la función 50; si ya lo fue, genera el error 20.
9	Sí no ha sido activada antes en el bloque alguna otra función con parámetros, activa la función 92 ; si ya lo fue, genera el error 20.
10	Activa el indicador de salto de función (skip function).
11	Si está activa la variable del registro en memoria de paro exacto se desactiva, sino se activar.

continua en la siguiente página

Acciones de sintaxis

Num Acción	Descripción
12	Activa al grupo M al que pertenece la función con el número de M de la palabra y enciende el registro en memoria para generar el bloque de dicha función miscelánea.
13	Activa al grupo G al que pertenece la función con el número de G de la palabra.
14	Activa al grupo G al que pertenece la función con el número de G y enciende el registro en memoria para generar el bloque de dicha función preparatoria.
15	Activa la función para terminar el programa.

Cuando se cargó la información completa del bloque en análisis, se puede terminar la revisión de sintaxis de este. Se verifica que la estructura de datos que contiene los parámetros especificados en último bloque analizado y el estado de la memoria sean suficientes para realizar la operación deseada; si son insuficientes se genera un error. Estas operaciones se listan a continuación:

1. Si hay parámetros activos y no existe función en el bloque (o función modal especificada anteriormente) a la que puedan ser asociados, se genera un error. Los parámetros de entrada se asocian a las funciones como aparecen en la tabla 7.

2. Si hay una función G28 se activa y se almacena el punto asociado a ella.

3. Si hay una función G29; verifica que se haya activado antes una función G28 (no necesariamente en el bloque anterior), si no fue así genera error.

4. Para cualquier función que necesite parámetros especificados en el mismo bloque verifica que estos sean suficientes para poder generar el(los) bloque(s) de salida; si no lo son, genera error.

5. Se verifica que no se haya activado más de una función que requiere los mismos parámetros de entrada especificados en el bloque.

6. Si se especificó la función S y la función preparatoria G50, se establece el límite máximo de S para posteriores verificaciones.

7. Si se especificó S durante el bloque pero no para configurarla a través de G50, se verifica que el valor proporcionado en el bloque no exceda el límite anteriormente definido.

8. Sí se va a realizar alguna función de corte se verifica que: la mordaza esté cerrada, se haya definido antes el sentido de rotación del husillo, que el valor de F (la velocidad de alimentación) haya sido especificado antes o en el bloque actual y que se haya generado un bloque definiendo el valor máximo para S a través de la función G50.

Si no se generó algún error con alguna de las revisiones se termina la verificación de sintaxis del bloque de entrada.

4.3. Comprobación de tipos

En este compilador no existe propiamente una fase de análisis semántico pero se realizan revisiones de tipos. En la fase de léxico se verifica que las variables que solo puedan contener números enteros no acepten fracciones; durante la verificación de sintaxis, al asignar un valor a un parámetro se verifica que dicho valor esté dentro de los límites especificados a través de la tabla 6; los valores de esta tabla pueden ser modificados por el encargado de la máquina-herramienta. Para las funciones de corte o posicionamiento que utilizan los parámetros U o W, se calcula su equivalente en valor absoluto y se verifica que este no exceda los límites especificados a través de X o Z en la tabla.

La descripción de los campos de la tabla 6 son:

1. Nombre de la variable;

2. P/N, con un letra P para cuando la variable es de posicionamiento/corte, N cuando no lo es;

3. Número de Token, llave para relacionar la identificación hecha por léxico con las diferentes etapas del compilador

4. Límite Inferior, número que define el valor mínimo en centímetros si se trata de una variable de posicionamiento/corte ó límite inferior en las correspondientes unidades para cualquier otro dato.

5. Límite Superior, número que define el valor máximo válido en centímetros si es una variable de posicionamiento/corte ó límite superior para cualquier otro dato.

4.4. Detección e información de errores

Cada fase puede encontrar errores; al detectar un error hay que darle un tratamiento para poder continuar con la compilación, permitiendo así la verificación completa del programa fuente.

En este compilador en las fases de análisis sintáctico y generación de código se detectan la mayoría de los errores posibles en una compilación. La fase de léxico detecta los errores donde un conjunto de los caracteres en un bloque no forman algún componente válido del lenguaje. Los errores donde la secuencia de componentes léxicos o palabras en un bloque violan las reglas de estructura del lenguaje son determinados por la fase de análisis sintáctico. En la fase de generación de código se detectan aquellos errores que relacionan más de un bloque de entrada para generar uno o varios bloques de salida.

Por la naturaleza del lenguaje G, para dar tratamiento a los errores, se interrumpe la verificación del bloque al detectar el primer error en el mismo.

Durante la compilación se crea una lista de los errores que contiene la siguiente información:

1. El número de error
2. La descripción del error
3. La línea donde comenzó el error
4. La columna donde inició el token en donde fue detectado
5. Para los errores de léxico, la palabra en donde se detectó el error.

Al final del proceso de compilación se muestra una caja de mensaje con la cantidad de errores encontrados en las diferentes etapas y una hoja en la parte inferior de la pantalla con la descripción de cada uno de los errores. En la tabla 5 aparece una lista con los errores que pueden ser detectados por este compilador.

4.5. Generación de código

Esta fase genera secuencias de instrucciones de máquina que ejecutan las tareas indicadas en un bloque a partir de la estructura de datos proveniente del bloque analizado y

del estado de la memoria del compilador resultado de los bloques anteriores; se analiza la información y se produce una, más de una, o ninguna de las siguientes directivas de salida, en el orden en que aparecen:

1. La función S (velocidad del husillo) si fue especificada sin G50 ó G97.
2. Cualquier cambio de herramienta.
3. Las funciones misceláneas activadas que deben ser enviadas al inicio del bloque; en la tabla 4, estas funciones contienen un número uno en la segunda columna.
4. Las funciones preparatorias que generan bloque de salida pero que no son de movimiento o de corte.
5. La función de posicionamiento o corte.
6. Las funciones misceláneas que se aplican durante o al final del bloque, estas contienen un número diferente de uno en la segunda columna en la tabla 4.

Las **funciones preparatorias** que generan salida con parámetros aparecen en la tabla 7 y su contenido se describe a continuación:

1. Para la función G00 se especifican los incrementos con signo en cuentas de encoder de la última a la nueva posición definida en el bloque de entrada para cada uno de los ejes.

   Las directivas de esta instrucción para su ejecución en la máquina-herramienta son:

VS FeedrateMax	Especifica la velocidad del vector en una secuencia de movimiento.
VP Inc X, Inc Z	Define las coordenadas objetivo de un segmento de línea recta en una secuencia de movimiento de dos ejes.
BGS	Comienza la secuencia.

   Donde:

   FeedrateMax es un número que indica la velocidad máxima de alimentación.
   Inc X, Inc Z son dos números separados por coma para indicar el incremento en cuentas de encoder en el eje X y Z respectivamente.

2. Para la función G01 también se especifican los incrementos con signo en cuentas de encoder en los ejes y la velocidad de alimentación definida para el corte.

VS Feedrate	Especifica la velocidad del vector en una secuencia de movimiento.
VP Inc X, Inc Z	Define las coordenadas objetivo de un segmento de línea recta en una secuencia de movimiento de dos ejes.
BGS	Comienza la secuencia.

Donde:

Feedrate es el número que indica la velocidad de alimentación para la interpolación lineal en proceso, siempre menor o igual a FeedrateMax.

Inc X, Inc Z son dos números separados por coma para indicar el incremento en cuentas de encoder en el eje X y Z respectivamente.

3. Para G02 y G03 se proporciona el punto final de la interpolación con incrementos en cuentas de encoder para cada uno de los ejes, el ángulo que forma el punto actual con respecto al origen de la máquina, el incremento del ángulo en radianes con signo para llegar al punto final de la interpolación y la velocidad de alimentación definida para el corte.

   Si es una interpolación circular en sentido horario (G02) o antihorario (G03), esta, queda explícita en el sentido de los ángulos especificados. La interpolación circular puede estar compuesta también de dos grupos de salida, esto depende de sí los parámetros del arco de entrada son exactos, si es así, será solo un grupo; si no lo son y ocurre que el centro del arco fue especificado a través de los parámetros I,K y el punto final del arco no es equidistante del centro de este, se generan dos grupos: el primero de ellos para una interpolación circular del punto actual al punto formado por la intersección de la circunferencia con el centro dado por I,K y la recta que va del centro de circunferencia al punto final especificado y el segundo bloque para la interpolación del punto de intersección al punto final.

   Cuando la especificación del centro del arco se realiza a través de los parámetros (i, k) y los puntos de inicio y fin no forman parte del mismo arco, usualmente hay un error en los sistemas tradicionales, en este sistema, como lo comenta Smid acerca de los sistemas modernos, el cálculo se realiza de la siguiente manera:

   Se desea ir del punto actual $P_1 = (x_1, z_1)$ al punto $P_2 = (x_2, z_2)$ formando un arco con un centro $c = (x_c, z_c) = (x_1 + i, z_1 + k)$
   el radio de la circunferencia está dado por $r = \sqrt{i^2 + k^2}$
   y la distancia d del punto c al punto P_2 por $d = \sqrt{(z_2 - z_c)^2 + (x_2 - x_c)^2}$
   Si $r \neq d$ y $m = \arctan((z_2 - z_c)/(x_2 - x_c))$

la constante de la recta que se forma del punto c al punto P_2 es
$bRect = z_2 - mx_2$ la ecuación de la recta $z_r = mx_r + bRect$ en intersección con la circunferencia con centro en c da como resultado:

$$(m^2 + 1){x_f}^2 + (2m(bRect - z_c) - 2x_c)x_f + (bRect^2 - z_c)^2 - r^2 = 0$$

si denotamos:

$$\begin{aligned} a &= m^2 + 1 \\ b &= 2m(bRect - z_c) - 2x_c \\ c &= (bRect^2 - z_c)^2 - r^2 \end{aligned}$$

tenemos entonces:

$$\begin{aligned} xf_1 &= \frac{-b + \sqrt{b^2 - 4ac}}{2a} \\ xf_2 &= \frac{-b - \sqrt{b^2 - 4ac}}{2a} \end{aligned}$$

Si se cumple que $x_2 \geq xf_1 \geq x_c$ hacemos $x_f = xf_1$, de otra forma $x_f = xf_2$

$$\begin{aligned} z_f &= mx_f + bRect \\ \theta &= \frac{180(arctan\frac{z_f}{x_f})}{\pi} \\ \Delta\theta &= \frac{180(arctan(\frac{z_2 - z_1}{x_2 - x_1}))}{\pi} \end{aligned}$$

Si la interpolación circular es en sentido anti-horario hacemos$\Delta\theta = -\Delta\theta$.

Entonces las directivas de salida cuando son dos grupos se forman de la siguiente manera:

VS Feedrate	Especifica la velocidad del vector en una secuencia de movimiento.
CR r,θ, $\Delta\theta$	Define un segmento de arco de dos dimensiones.
BGS	Comienza la secuencia.
VS Feedrate	Especifica la velocidad del vector en una secuencia de movimiento.
VP Inc X, Inc Z	Define las coordenadas objetivo de un segmento de línea recta en una secuencia de movimiento de dos ejes.
BGS	Comienza la secuencia.

Donde:

Feedrate es el número que indica la velocidad de alimentación para la interpolación circular en proceso, siempre menor o igual a FeedrateMax.

r radio del segmento de arco, número real sin signo.

θ ángulo de inicio, número real con signo.

$\Delta\theta$ incremento del ángulo, número real con signo.

Si por las características del punto destino especificadas en la interpolación circular solo se va a generar un grupo de instrucciones de salida, estas quedan definidas por las primeras tres directivas.

4. Para G04 se describe el valor de la pausa en milisegundos a través de la instrucción:

WT n

Donde:

n es un número entero que indica la cantidad en milisegundos de la pausa.

5. Para G28 se generan grupos de bloques de salida con parámetros de incremento en cuentas de encoder en cada eje; el primer bloque es para realizar el desplazamiento del punto actual al punto intermedio especificado por la instrucción de entrada y el segundo bloque para el movimiento del punto intermedio al origen de la máquina a través de dos grupos de salida compuestos por las mismas directivas que G00 pero para dos movimientos diferentes, uno grupo seguido del otro.

6. Para G29 también se generan dos grupos de directivas de salida con parámetros de incremento en cuentas de encoder como en G28; solo que, el primer grupo es para realizar el desplazamiento del punto actual al especificado por la instrucción de entrada y el segundo para el movimiento del punto intermedio al especificado a través de la función G28 relacionada con el G29 en proceso. Al igual que G28 las directivas son como en G00 para los dos movimientos.

Para las funciones G50 y G92 se especifica el valor de S como la cantidad máxima de revoluciones por minuto, solo utilizado por el compilador, no se realiza función de salida.

Para G97 se configura la entrada en revoluciones por minuto, cancelando así G96.

También existen otras funciones G que le indican al controlador la activación o desactivación de alguna función de la máquina-herramienta que requieren que se genere su bloque de salida, aunque no necesitan parámetros; las funciones G40, G41, G42 y G96 son de este tipo.

Para la **velocidad del husillo** se utiliza la función Offset.

OF „ n

Donde:

n es un número con signo para indicar un valor en volts.

Para realizar los cambios de **herramienta** se genera una llamada a una función.

R n

Donde:

n es un número de herramienta a cargar.

Se conoce que las **funciones misceláneas** representan la activación o desactivación de un dispositivo en la máquina-herramienta, para aquellas que se mencionan en la tabla 4 se generan directivas de uno o más bloques de activación o desactivación, según sea el caso. En la tabla se muestra una lista de las definidas hasta el momento. La salida en la tarjeta es a través de una palabra de ocho bits en donde CB indica inicializar y SB encender el bit.

Tabla 4.3: Salida para algunas funciones misceláneas

Función	Salida
M03	CB4,SB3
M04	CB3,SB4
M05	CB1
M08	SB5
M09	CB5
M10	SB2
M11	CB2
M30	CB1,CB2,CB3,CB4,CB5

4.6. Ventajas del desarrollo

Al utilizar este desarrollo se tienen muchas ventajas. Sabemos que la compilación verifica que el programa esté libre de errores antes de ser ejecutado por la máquina-herramienta, esto facilita su depuración y asegura menor cantidad errores al momento de maquinar una pieza, además las validaciones, como por ejemplo los límites geométricos para cada eje o los valores máximos y mínimos que pueden tomar los diferentes parámetros, proveen seguridad al operador y pueden evitar colisiones en la máquina-herramienta.

Capítulo 5

Pruebas y Conclusiones

5.1. Ubicación del desarrollo en los diferentes modos de operación del CNC

El sistema CNC del que forma parte este trabajo consta de 6 modos, para seleccionar cada uno de ellos se utiliza la perilla del gabinete del CNC como se muestra en la figura 5.1.

Los modos de operación son:

EDIT MEMORY MDI TAPE JOG HANDLE

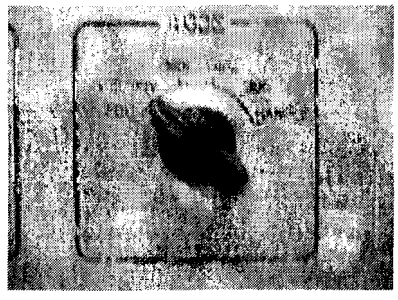


Figura 5.1: Perilla de gabinete.

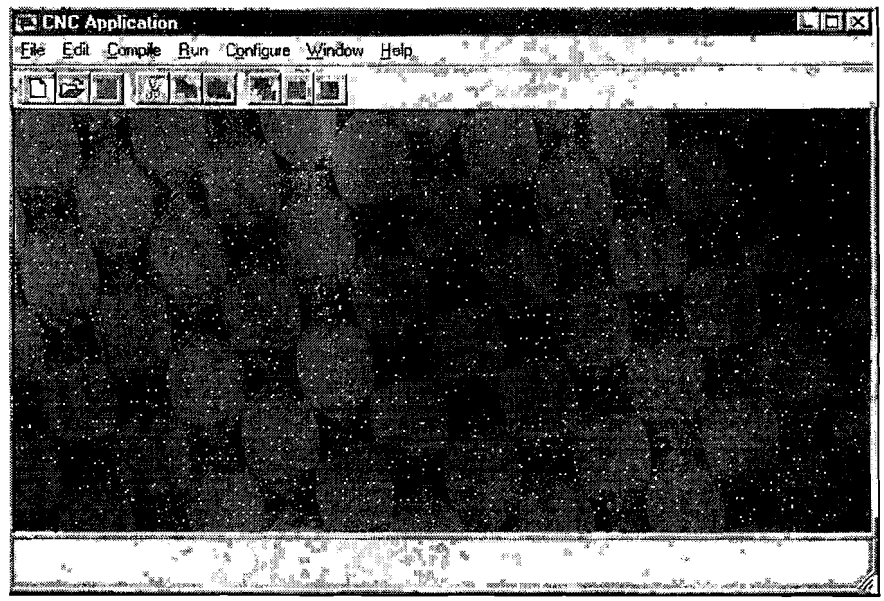


Figura 5.2: Pantalla principal de la aplicación.

Este proyecto contiene el desarrollo de la opción **Edit** y la generación de las directivas de la opción **Memory** hasta antes de que estas sean ejecutadas por la máquina-herramienta.

5.2. EDIT

Al seleccionar a través de la perilla la opción Edit, aparece el sistema desarrollado en este trabajo con la fig. 5.2 como pantalla principal.

La aplicación tiene los menús File, Edit, Compile, Run, Configure, Window y Help.

A través del menú **File** se administran los programas para el CNC, se pueden cargar archivos de programa, crear nuevos, grabar, grabar como, e imprimir. Se utiliza una ventana de diálogo para visualizar los programas disponibles.

Una vez que se crea o carga un programa, el contenido de este se puede modificar a través del editor del sistema que proporciona las operaciones básicas de edición utilizando el portapapeles: cortar, copiar, pegar, seleccionar todo o deshacer; las opciones de edición se pueden ejecutar de manera automática a través de los conjuntos de teclas tradicionales Ctrl-X, Ctrl-C, Ctrl-V, Ctrl-Z o en el menú **Edit**.

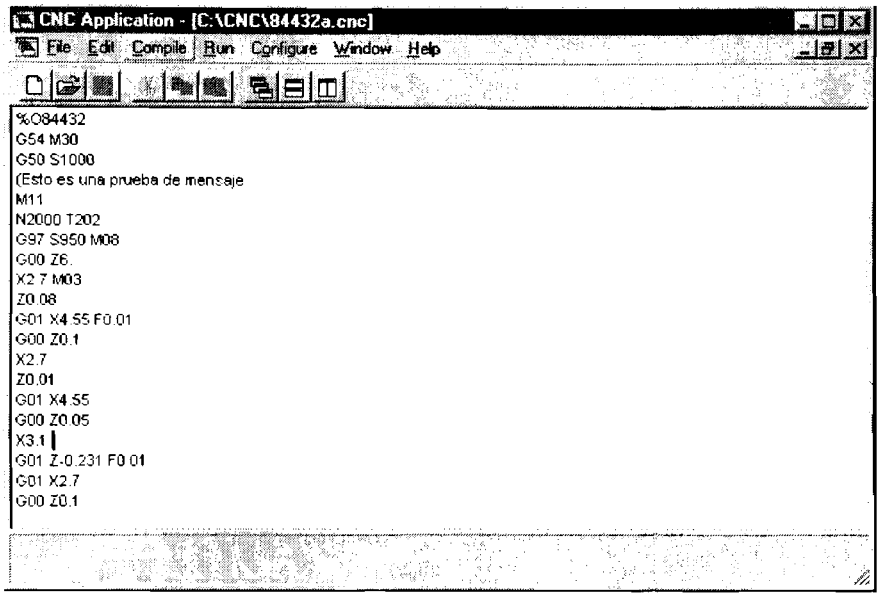


Figura 5.3: Editor de la aplicación.

El menú **Compile** se usa para realizar el proceso de compilación del programa activo; si no encuentran errores durante la compilación, se genera un programa objeto y se despliega una caja de diálogo como en 5.4, indicando una compilación exitosa; si hubo al menos un error, se indica la cantidad de errores encontrados y se muestra una lista en la parte inferior de la pantalla con campos que los describen; la información en la lista es: número de error, descripción columna y línea donde tuvo inicio. Los errores pueden ser corregidos a través de la misma pantalla de edición.

En la figura 5.5 se puede ver el resultado de la compilación del programa activo (su marco se encuentra resaltado); cuando se pulsa el botón *OK*, se despliega en la parte inferior de la pantalla la lista de los errores encontrados, como en 5.6.

El menú **Run** tiene dos opciones, ejecutar el programa paso a paso o de inicio a fin.

El menú **Configure** permite modificar algunos de los parámetros que utiliza el compilador para sus validaciones como los valores máximos y mínimos que pueden tomar cada uno de los ejes para sus movimientos, los 20 offsets de trabajo, los 20 offsets de la herramienta, etc.

A través del menú Window se puede modificar la forma de visualizar las pantallas de edición abiertas: de forma horizontal, vertical o en forma de cascada.

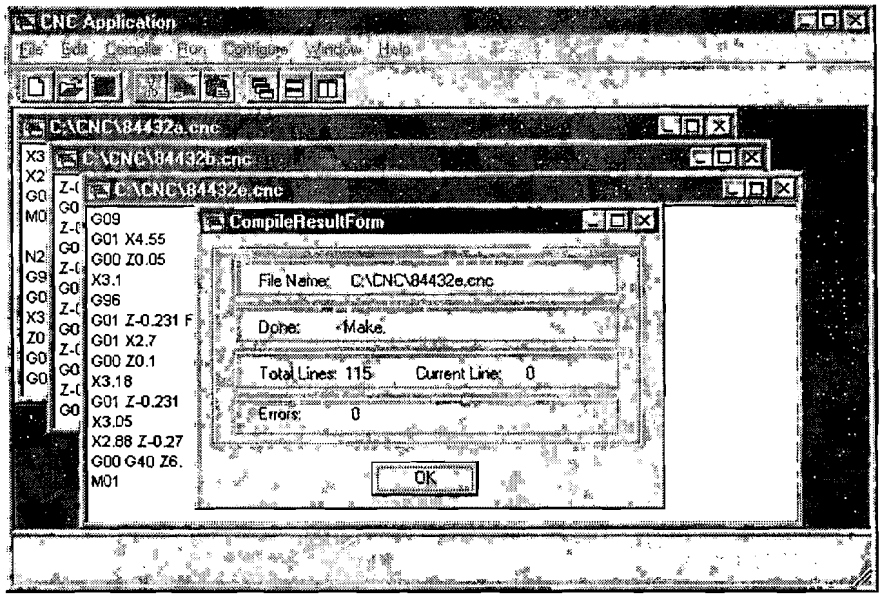


Figura 5.4: Dialogo de compilación de un programa sin errores.

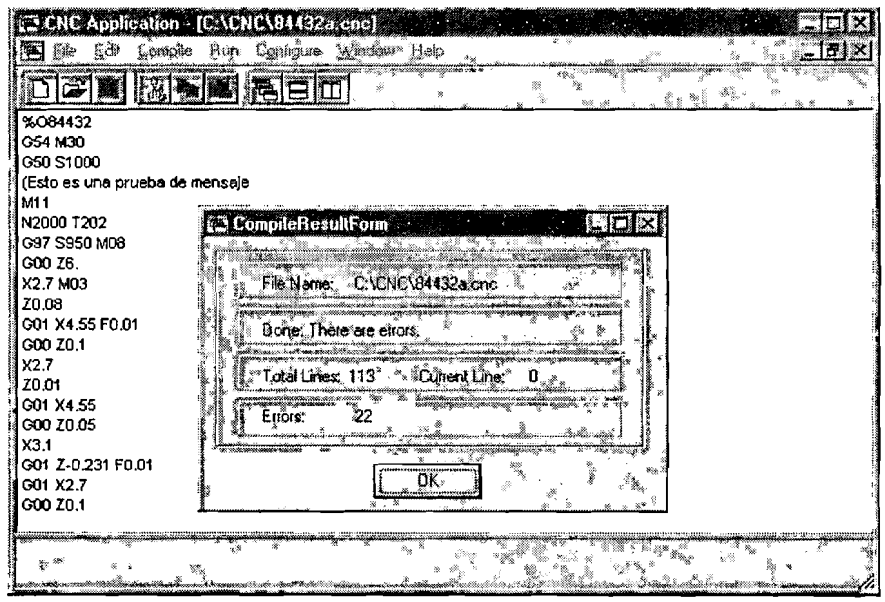


Figura 5.5: Dialogo de compilación de un programa con errores.

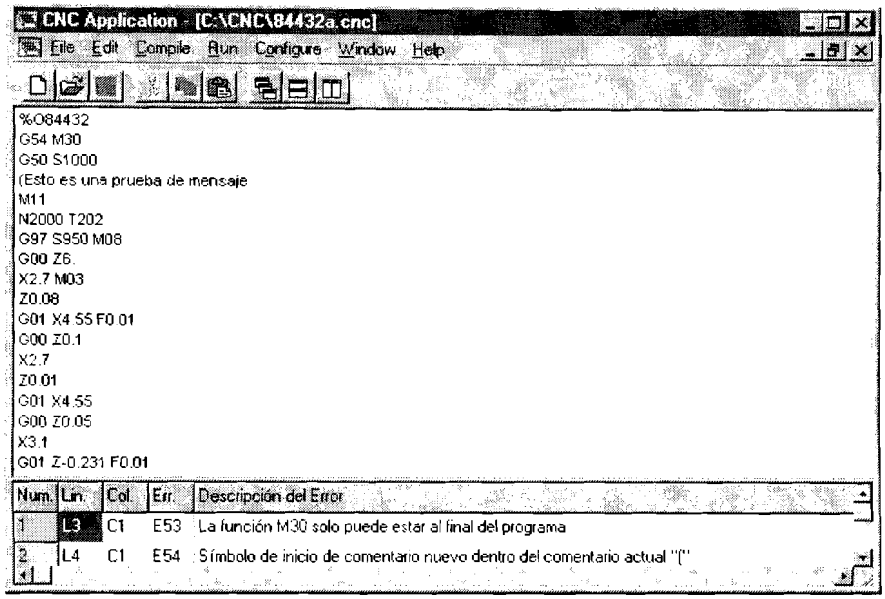


Figura 5.6: Programa compilado con su lista de errores detectados.

5.3. MEMORY

Sí a través de la perilla se elige la opción Memory, el programa activo se puede ejecutar de dos diferentes modos: Continue o Trace (modo continuo o modo paso a paso). El trabajo para la ejecución de las directivas a través de la tarjeta fue desarrollado por el Dr. Herrera y se puede consultar en [Herrera, 2004].

Durante la ejecución de cualquier programa se puede observar por medio del sistema la posición en que se encuentra la herramienta, la instrucción en ejecución, la velocidad de alimentación, y la velocidad y sentido de giro del husillo (CW) o (CCW).

5.4. Prueba de maquinado de una parte

La finalidad de esta prueba fue comprobar que un programa en código objeto generado por una compilación a través de este sistema puede ser ejecutado exitosamente en un torno a través de una tarjeta Galil. El método para realizar esto fue tomar un programa fuente, compilarlo y enviar su programa objeto a la máquina-herramienta para verificar la ejecución adecuada en el maquinado de la pieza.

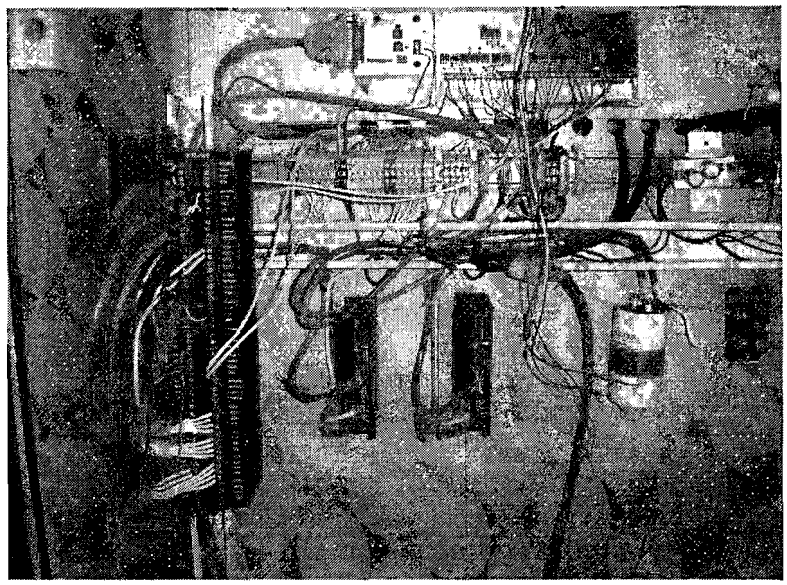


Figura 5.7: Fotografía de conexión entre torno y computadora.

Se tomó un programa en código G y se procesó a través de un simulador realizado por el Dr. Herrera en el Instituto Tecnológico de Estudios Superiores de Monterrey para máquinas EMCO; se realizó la simulación del maquinado de una pieza de un alfil de juego de ajedrez. El programa utilizado a través del simulador se muestra en la parte izquierda de la tabla 5.1. La figura 5.8 se obtuvo en pantalla como resultado de la simulación.

Tabla 5.1: Programas en código G

Programa para máquina EMCO	Programa bajo estándar ISO
120.00 22.00 5.00 5.00	(120.00 22.00 5.00 5.00)
N00 G92 X+3234 Z+00500 F—	N100 G00 X+3234 Z+00500
	N120 G00 X+2234 Z+00100
N01 G90	G50 S5000 M11 M03 F150
N02 G00 X+2234 Z+00100 F—	
N03 G84 X+2034 Z-04500 F150	N130 G00 X+2034
	N135 G01 Z-04500
	N137 G00 X+2234 Z+00100
N04 G84 X+1934 Z-03550 F150	N140 G00 X+1934
	N145 G01 Z-03550

continua en la siguiente página

Tabla 5.1: Programas en código G

Programa para máquina EMCO	Programa bajo estándar ISO
	N147 G00 X+2234 Z+00100
N05 G84 X+1834 Z-03500 F150	N150 G00 X+1834
	N155 G01 Z-03500
	N157 G00 X+2234 Z+00100
N06 G84 X+1634 Z-03487 F150	N160 G00 X+1634
	N165 G01 Z-03487
	N167 G00 X+2234 Z+00100
N07 G84 X+1434 Z-03446 F150	N170 G00 X+1434
	N175 G01 Z-03446
	N177 G00 X+2234 Z+00100
N08 G84 X+1374 Z-00593 F150	N180 G00 X+1374
	N185 G01 Z-00593
	N187 G00 X+2234 Z+00100
N09 G84 X+1234 Z-00573 F150	N190 G00 X+1234
	N195 G01 Z-00573
	N197 G00 X+2234 Z+00100
N10 G84 X+1134 Z-00532 F150	N200 G00 X+1134
	N205 G01 Z-00532
	N207 G00 X+2234 Z+00100
N11 G84 X+1034 Z-00400 F150	N210 G00 X+1034
	N215 G01 Z-00400
	N217 G00 X+2234 Z+00100
N12 G84 X+0934 Z-00100 F150	N220 G00 X+0934
	N225 G01 Z-00100
	N227 G00 X+2234 Z+00100
N13 G01 X+0800 Z000000 F150	N230 G01 X+0800 Z000000
N14 G03 X+1000 Z—— F050	N240 G03 X+1000 R1400 F050
N15 G01 X+1000 Z-00400 F050	N250 G01 X+1000 Z-00400
N16 G02 X+1400 Z—— F051	N260 G02 X+1400 R1400 F051
N17 G01 X+1400 Z-01100 F050	N270 G01 X+1400 Z-01100 F050
N18 G01 X+1200 Z-01300 F150	N280 G01 X+1200 Z-01300 F150
N19 G01 X+1200 Z-03400 F150	N290 G01 X+1200 Z-03400
N20 G01 X+1200 Z-01300 F150	N300 G01 X+1200 Z-01300
N21 G01 X+1100 Z-01450 F150	N310 G01 X+1100 Z-01450

continua en la siguiente página

Tabla 5.1: Programas en código G

Programa para máquina EMCO	Programa bajo estándar ISO
N22 G01 X+1100 Z-03310 F150	N320 G01 X+1100 Z-03310
N23 G01 X+1100 Z-01450 F150	N330 G01 X+1100 Z-01450
N24 G01 X+1034 Z-01600 F150	N340 G01 X+1034 Z-01600
N25 G01 X+1034 Z-03100 F150	N350 G01 X+1034 Z-03100
N26 G00 X+1800 Z000000 F—	N360 G00 X+1800 Z000000
N27 G00 X+1800 Z-01100 F—	N370 G00 X+1800 Z-01100
N28 G01 X+1400 Z-01100 F150	N380 G01 X+1400 Z-01100 F150
N29 G01 X+1000 Z-01600 F050	N390 G01 X+1000 Z-01600 F050
N30 G01 X+1000 Z-03100 F050	N400 G01 X+1000 Z-03100
N31 G02 X+1800 Z—— F050	N410 G02 X+1800 R1400
N32 G01 X+2000 Z-03600 F050	N420 G01 X+2000 Z-03600
N33 G01 X+2000 Z-03850 F050	N430 G01 X+2000 Z-03850
N34 G01 X+1800 Z-04050 F050	N440 G01 X+1800 Z-04050
N35 G01 X+2000 Z-04250 F050	N450 G01 X+2000 Z-04250
N36 G01 X+2000 Z-04500 F050	N460 G01 X+2000 Z-04500
N37 G00 X+3234 Z+00500 F—	N470 G00 X+3234 Z+00500
N38 G22 X— Z—— F—	N480 M30

Al programa procesado a través del simulador se le agregaron funciones al inicio del programa para definir el sentido y la velocidad máxima de rotación del husillo y para cerrar la mordaza. En código para máquinas EMCO la función G84 X.. Z.. es un ciclo de torneado que para el estándar ISO utilizado equivale a realizar un posicionamiento rápido en el valor especificado para el eje X, una interpolación lineal en el valor dado para Z y un posicionamiento rápido en los valores del punto de partida de X y Z; así, se trasformó cada una de las instrucciones G84 encontradas en el programa. También se cambió la sintaxis de las interpolaciones circulares, y por último, se colocó una función M30 equivalente al G22 de EMCO al final del programa.

Tabla 5.2: Programa Objeto Resultado

```
VS 20000
VP 15149,19250
BGS
VS 20000
```

continua en la siguiente página

Tabla 5.2: Programa Objeto Resultado

```
VP 1500,600
BGS
CB2
CB4
SB3
VS 20000
VP 300,0
BGS
VS 3000
VP 0,6900
BGS
VS 20000
VP -300,-6900
BGS
VS 20000
VP 450,0
BGS
VS 3000
VP 0,5475
BGS
VS 20000
VP -450,-5475
BGS
VS 20000
VP 600,0
BGS
VS 3000
VP 0,5400
BGS
VS 20000
VP -600,-5400
BGS
VS 20000
VP 900,0
BGS
```

continua en la siguiente página

Tabla 5.2: Programa Objeto Resultado

VS 3000
VP 0,5381
BGS
VS 20000
VP -900,-5381
BGS
VS 20000
VP 1200,0
BGS
VS 3000
VP 0,5319
BGS
VS 20000
VP -1200,-5319
BGS
VS 20000
VP 1290,0
BGS
VS 3000
VP 0,1040
BGS
VS 20000
VP -1290,-1040
BGS
VS 20000
VP 1500,0
BGS
VS 3000
VP 0,1010
BGS
VS 20000
VP -1500,-1010
BGS
VS 20000
VP 1650,0
continua en la siguiente página

Tabla 5.2: Programa Objeto Resultado

```
BGS
VS 3000
VP 0,948
BGS
VS 20000
VP -1650,-948
BGS
VS 20000
VP 1800,0
BGS
VS 3000
VP 0,750
BGS
VS 20000
VP -1800,-750
BGS
VS 20000
VP 1950,0
BGS
VS 3000
VP 0,300
BGS
VS 20000
VP -1950,-300
BGS
VS 3000
VP 2151,150
BGS
VS 1000
CR 300,46.8000,0.0000
BGS
VS 1000
VP 0,600
BGS
VS 1020
```

continua en la siguiente página

Tabla 5.2: Programa Objeto Resultado

```
CR 600,48.1000,0.0000
BGS
VS 1000
VP 0,1050
BGS
VS 3000
VP 300,300
BGS
VS 3000
VP 0,3150
BGS
VS 3000
VP 0,-3150
BGS
VS 3000
VP 150,225
BGS
VS 3000
VP 0,2790
BGS
VS 3000
VP 0,-2790
BGS
VS 3000
VP 99,225
BGS
VS 3000
VP 0,2250
BGS
VS 20000
VP -1149,-4650
BGS
VS 20000
VP 0,1650
BGS
```

continua en la siguiente página

Tabla 5.2: Programa Objeto Resultado

```
VS 3000
VP 600,0
BGS
VS 1000
VP 600,750
BGS
VS 1000
VP 0,2250
BGS
VS 1000
CR 1200,53.1000,0.0000
BGS
VS 1000
VP -300,750
BGS
VS 1000
VP 0,375
BGS
VS 1000
VP 300,300
BGS
VS 1000
VP -300,300
BGS
VS 1000
VP 0,375
BGS
VS 20000
VP -1851,-7500
BGS
```

El programa en código fuente cargado para el compilador se muestra en la parte derecha de tabla 5.1; se procesó la compilación y se obtuvo el código objeto indicado en la tabla 5.2.

En el resultado se puede apreciar la activación o desactivación de algunas señales dig-

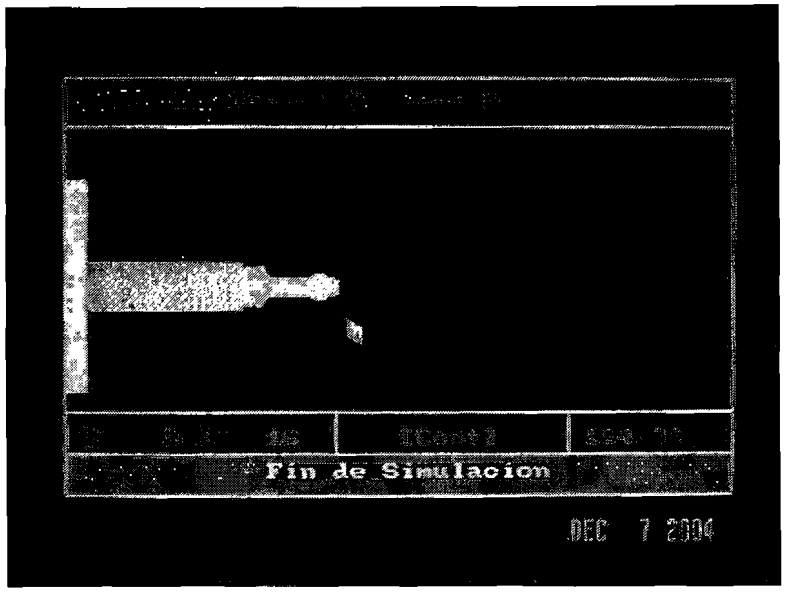


Figura 5.8: Fotografía del resultado en el simulador.

itales (CB2, CB4, SB3, etc), estas controlan funciones como el sentido en que gira el husillo, el encender o apagar el refrigerante, el abrir o cerrar la mordaza, etc.

Este código se proceso en la máquina-herramienta a través de la tarjeta del controlador utilizando el trabajo del Dr. Herrera citado en [Herrera, 2004] y se obtuvo la pieza que se muestra en la figura 5.10.

5.5. Conclusiones

El sistema funcionó correctamente, una vez que se hizo la prueba del maquinado de la pieza, se pudo observar que el proceso de transformación del código fuente en el código objeto para la máquina-herramienta fue exitoso así como la activación o desactivación de las diferentes salidas manipuladas a través de las funciones misceláneas. El trabajo cumple su objetivo y además permite validar la información del programa fuente obteniendo un proceso de maquinado más seguro para el operador y para evitar daños a la maquina o a las herramientas.

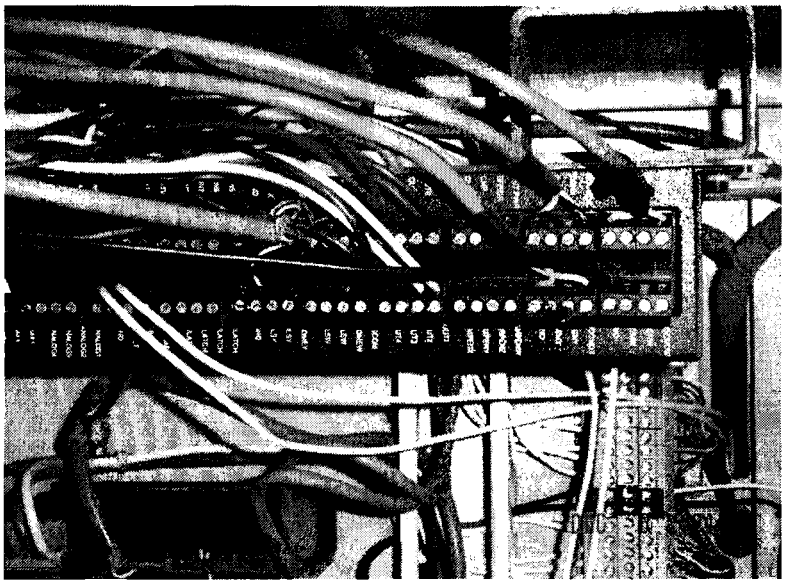


Figura 5.9: Switches que se manejan a través de funciones misceláneas.

Figura 5.10: Fotografía de la pieza maquinada.

Figura 5.11: Fotografía del torno utilizado.

Bibliografía

(Aho, Sethi y Ullman, 1986) Aho A., Sethi R. y Ullman J. "Compilers-Principles, Techniques and Tools". Adison Wesley, 1986.

(Aho, Sethi y Ullman, 1972) Aho A., Sethi R. y Ullman J. "The Theory of Parsing, Traslation and Compiling". Prentice Hall, 1972.

(Alaniz, 2003) Alaniz, Pedro. Instrumentación y Control de una Máquina Herramienta de Dos Ejes, Tesis de Maestría en Ciencias en Instrumentación y Control Automático, Facultad e Ingeniería, Universidad Autónoma de Querétaro, Querétaro, México, 2003.

(Altintas, 2000) Altintas, Y. "Manufacturing Automation". Cambridge University Press, 2000.

(Canses y Màrquez, 2002) Canses Muñoz R. y Màrquez Villodre L. "Lenguajes, gramáticas y autómatas". Alfaomega 2002

(Herrera, 2004) Herrera, Gilberto. Sistema de Control para Máquinas-Herramienta CHROM-II, Certificado de derechos de autor, Instituto Nacional de Derechos de Autor, 2004.

(Herrera, 1992) Herrera, Gilberto. A Modular CNC Controller, Dissertation Submitted for the Doctoral Degree in Engineering, Computer and Automation Institute, Hungarian Academy of Sciences, Department of Manufacturing Engineering, Technical University of Budapest, Budapest, Hungary, 1992.

(Seames, 1990) Seames, W. Computer Numerical Control. DELMAR, 1990.

(Smid, 2003) Smid, Peter. CNC Programming Handbook, Industrial Press, Inc., NY, USA, 2003.

(Srikant y Shankar, 2003) Editado por Srikant Y.N., Shankar P. "The Compiler Design Handbook". CRC PRESS, 2003.

(Wolfe, 1996) Wolfe Michael. "High performance compilers for parallel computing". Adison Wesley, 1996.

Tabla de Funciones Preparatorias

Tabla 3: Funciones Preparatorias

Gr.	Nom.	Núm.	Desc.Inglés	Desc.Español
01	G00	300	Rapid positioning	Posicionamiento rápido
01	G01	301	Linear interpolation	Interpolación lineal
01	G02	302	Circular interpolation clockwise	Interpolación circular en sentido horario
01	G03	303	Circular interpolation counterclockwise	Interpolación circular en sentido antihorario
00	G04	304	Dwell	Tiempo de espera
00	G09	309	Exact stop check	Señal de paro exacto
06	G20	320	English units of input	Unidades de entrada en sistema inglés
06	G21	321	Metric units of input	Unidades de entrada en sistema métrico
00	G28	328	Machine zero return	Regreso al punto de referencia
00	G29	329	Return from machine zero	Regreso desde el punto de referencia
00	G31	331	Skip function	Función de salto
07	G40	340	Tool nose radius offset cancel	Cancela offset de radio de la herramienta
00	G50	350	Tool positioning register / Maximum r/min preset	Limitación de la velocidad máxima del husillo
12	G54	354	Work coordinate offset 1	Coordenada límite de trabajo 1
12	G55	355	Work coordinate offset 2	Coordenada límite de trabajo 2
12	G56	356	Work coordinate offset 3	Coordenada límite de trabajo 3

continua en la siguiente página

Tabla 3: Funciones Preparatorias

Gr.	Nom.	Núm.	Desc.Inglés	Desc.Español
12	G57	357	Work coordinate offset 4	Coordenada límite de trabajo 4
12	G58	358	Work coordinate offset 5	Coordenada límite de trabajo 5
12	G59	359	Work coordinate offset 6	Coordenada límite de trabajo 6
03	G90	390	Absolute command	Programación absoluta
03	G91	391	Incremental command	Programación incremental
00	G92	392	Tool position, register	Limitación de velocidad máxima del husillo
05	G94	394	Feedrate per minute	Velocidad de alimentación por minuto
05	G95	395	Feedrate per revolution	Velocidad de alimentación por revolución
17	G96	396	Constant surface speed mode (CSS)	
17	G97	397	Direct r/min input (cancela modo CSS)	Entrada directa de revoluciones por minuto
10	G98	398	Feedrate per minute	Velocidad de alimentación por minuto
10	G99	399	Feedrate per revolution	Velocidad de alimentación por revolución

Tabla de Funciones Misceláneas

Tabla 4: Funciones Misceláneas

Gr.	Ap.	Nom	Num	Desc.Inglés	Desc.Español
01	99	M00	600	Compulsory program stop	Paro programado
01	99	M01	601	Optional program stop	Paro opcional
01	99	M02	602	End of program (usually with reset, no rewind)	Fin de programa
02	01	M03	603	Spindle rotation normal	Husillo conectado en sentido horario
02	01	M04	604	Spindle rotation reverse	Husillo conectado en sentido antihorario
02	99	M05	605	Spindle stop	Paro de husillo
03	01	M07	607	Coolant mist ON	Mezcla de refrigerante conectada
03	01	M08	608	Coolant ON (coolant pump motor ON)	Flujo de refrigerante conectado
03	99	M09	609	Coolant OFF (coolant pump motor OFF)	Mezcla o flujo de refrigerante desconectado
04	00	M10	610	Chuck open	Mordaza abierta
04	00	M11	611	Chuck close	Mordaza cerrada
04	00	M12	612	Tailstock quill IN	Movimiento del cabezal móvil atrasado
04	00	M13	613	Tailstock quill OUT	Movimiento del cabezal móvil adelantado
04	00	M17	617	Turret indexing forward	
04	00	M18	618	Turret indexing reverse	
02	00	M19	619	Spindle orientation (optional)	Orientación del husillo

continua en la siguiente página

Tabla 4: Funciones Misceláneas

Gr.	Ap.	Nom	Num	Desc.Inglés	Desc.Español
04	00	M21	621	Tailstock forward	
04	00	M22	622	Tailstock backward	
05	00	M23	623	Thread gradual pull-out ON	
05	00	M24	624	Thread gradual pull-out OFF	
01	00	M30	630	Program end (always with reset and rewind)	Fin de programa con regreso al principio del programa
00	01	M31	631		
06	00	M41	641	Low gear selection	Selección de engrane bajo
06	00	M42	642	Medium gear selection 1	Selección 1 de engrane medio
06	00	M43	643	Medium gear selection 2	Selección 2 de engrane medio
06	00	M44	644	High gear selection	Selección de engrane alto
07	00	M48	648	Feedrate override cancel OFF (deactivated)	
07	00	M49	649	Feedrate override cancel ON (activated)	
08	00	M98	698	Subprogram call	Llamada a subprograma
08	00	M99	699	Subprogram end	Fin de subprograma

Tabla de Errores de Compilación

Tabla 5: Errores durante la Compilación

Err.	Descripción
1	Esperando número de función preparatoria
2	Esperando número de función miscelánea
3	Error de léxico
4	Función indefinida
5	No se encontró fin de comentario
6	Esperando número para descripción de parámetro
7	Parámetro definido más de una ocasión
8	Parámetro demasiado pequeño
9	Parámetro demasiado largo
10	Error de sintaxis
11	Función preparatoria no definida
12	Función miscelánea no definida
13	Esperando número entero
14	Esperando número entero o de punto flotante
15	Parámetro fuera de rango
16	Definición de otra función en proceso
17	No se puede definir G04 en la misma línea que otra función con parámetros
18	No se puede definir G28 en la misma línea que otra función con parámetros
19	No se puede definir G29 en la misma línea que otra función con parámetros
20	No se puede definir G50 o G92 en la misma línea que otra función con parámetros
21	Parámetro X no permitido para esta función
22	Parámetro Z no permitido para esta función

continua en la siguiente página

Tabla 5: Errores durante la Compilación

Err.	Descripción
23	Parámetro U no permitido para esta función
24	Parámetro W no permitido para esta función
25	Parámetro R no permitido para esta función
26	Parámetro I no permitido para esta función
27	Parámetro K no permitido para esta función
28	Los parámetros X y U son excluyentes
29	Los parámetros Z y W son excluyentes
30	Esperando parámetro
31	Esperando parámetro S
32	No es posible asociar parámetro
33	Definiendo el nombre de programa más de una ocasión
34	Definiendo el nombre del programa después de iniciado el programa
35	Definiendo más de un G con parámetros en un solo bloque
36	Parámetro P no permitido para esta función
37	Es necesario definir sentido de rotación para el husillo antes de corte
38	Es necesario M11 para realizar algún corte
39	Es necesario definir la velocidad de alimentación antes de un corte
40	Es necesario detener el husillo antes de abrir la mordaza
41	Es necesario especificar S a través de G50 antes de efectuar algún corte
42	No se ha activado G28 para la instrucción G29
43	El último número de línea proporcionado es mayor o igual a:
44	No se ha definido G para parámetro
45	No hay memoria suficiente para continuar
46	Número de herramienta no definido
47	OffSet de la herramienta fuera de rango
48	El valor de S excede el máximo especificado por G50
49	No existe parámetro para especificar el radio
50	Antes de definir S, es necesario especificar su valor máximo a través de G50
51	Necesario definir sentido de rotación del husillo
52	El programa debe ser terminado a través de la función M30
53	La función M30 solo puede estar al final del programa
54	Símbolo de inicio de comentario nuevo dentro del comentario actual

Otras tablas para el proceso de compilación

Tabla 6: Límites de Parámetros

Var	P/N	Ent/Dec	Número de Token	Lim. Inf	Lim. Sup.
X	P	D	124	-20000.07	20077.77777
Z	P	D	126	-30000.0008	880000.88888
S	N	E	119	1	1111
F	N	D	106	00.0001	1111
N	N	E	114	0	9999999
O	N	E	115	1	100000
P	N	E	116	1	10000

Tabla 7: Parámetros de Entrada

Func.Prep	X	Z	U	W	I	K	R	P
00	1	1	1	1	0	0	0	0
01	1	1	1	1	0	0	0	0
02	1	1	1	1	1	1	1	0
03	1	1	1	1	1	1	1	0
04	1	0	1	0	0	0	0	0
28	1	1	1	1	0	0	0	0
29	1	1	1	1	0	0	0	0
50	0	0	0	0	0	0	0	0
92	1	0	1	0	0	0	0	0

continua en la siguiente página

Tabla 7: Parámetros de Entrada

Func.Prep	X	Z	U	W	I	K	R	P
97	0	0	0	0	0	0	0	0

Tabla 8: Offsets de Herramienta

Offset en X	Offset en Y	Offset en Z
0	0	0
00.100	00.100	00.100
00.200	00.200	00.200
00.300	00.300	00.300
00.400	00.400	00.400
00.500	00.500	00.500
00.600	00.600	00.600
00.700	00.700	00.700
00.800	00.800	00.800
00.900	00.900	00.900
00.010	00.010	00.010
00.110	00.110	00.110
00.120	00.120	00.120
00.130	00.130	00.130
00.140	00.140	00.140
00.150	00.150	00.150
00.160	00.160	00.160
00.170	00.170	00.170
00.180	00.180	00.180
00.211	00.211	00.211

Tabla 9: Offsets de Trabajo

Offset en X	Offset en Y	Offset en Z
0	0	0
00.110	00.100	00.120
00.222	00.222	00.222
00.310	00.300	00.320

continua en la siguiente página

Tabla 9: Offsets de Trabajo

Offset en X	Offset en Y	Offset en Z
00.410	00.400	00.420
00.510	00.500	00.520
00.610	00.600	00.620
00.710	00.700	00.720
00.810	00.800	00.820
00.910	00.900	00.920
00.111	00.110	00.112
00.121	00.120	00.122
00.131	00.130	00.132
00.141	00.140	00.142
00.151	00.150	00.152
00.161	00.160	00.162
00.171	00.170	00.172
00.181	00.180	00.182
00.191	00.190	00.192
00.212	00.211	00.213

Tablas de Transición y Parseo

Tabla 10: Matriz de Parseo

	0	1	2	3	4	5	6	7
000	001,114,007,002	001,126,007,002	001,121,007,002	001,123,007,002	001,118,007,002	001,109,007,002	004,001,000,000	007,000,000,000
001	001,111,007,002	001,124,007,002	001,119,007,004	001,120,007,004	001,106,007,003	001,116,007,002	004,002,000,000	007,000,000,000
002	001,115,007,003	001,300,007,001	001,301,007,001	001,302,007,001	001,303,007,001	001,304,007,005	004,003,000,000	007,000,000,000
003	001,309,007,011	001,320,007,001	001,321,007,001	001,328,007,006	001,329,007,007	001,331,007,010	004,004,000,000	007,000,000,000
004	001,350,007,008	001,390,007,001	001,391,007,001	001,340,007,014	001,354,007,001	001,355,007,001	004,005,000,000	007,000,000,000
005	001,356,007,001	001,357,007,001	001,358,007,001	001,359,007,001	001,392,007,009	001,394,007,001	004,006,000,000	007,000,000,000
006	001,395,007,001	001,396,007,014	001,397,007,013	001,398,007,001	001,399,007,001	001,600,007,012	004,007,000,000	007,000,000,000
007	001,601,007,012	001,602,007,012	001,603,007,012	001,604,007,012	001,605,007,012	001,607,007,012	004,008,000,000	007,000,000,000
008	001,608,007,012	001,609,007,012	001,610,007,012	001,611,007,012	001,612,007,012	001,613,007,012	004,009,000,000	007,000,000,000
009	001,617,007,012	001,618,007,012	001,619,007,012	001,621,007,012	001,622,007,012	001,623,007,012	004,010,000,000	007,000,000,000
010	001,624,007,012	001,630,007,015	001,641,007,012	001,642,007,012	001,643,007,012	001,644,007,012	004,011,000,000	007,000,000,000
011	001,648,007,012	001,649,007,012	001,698,007,012	001,699,007,012	006,010,000,000	000,000,000,000	000,000,000,000	007,000,000,000

Tabla 11: Primera parte de Matriz de Transición

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
000	804	804	804	804	804	005	001	804	008	804	013	804	003	018	020	021	804	022	027	028	029
001	801	801	801	801	801	801	801	801	801	801	801	801	801	801	801	801	801	801	801	801	801
002	107	107	107	107	107	107	107	107	107	107	107	107	107	107	107	107	107	107	107	107	107
003	802	802	802	802	802	802	802	802	802	802	802	802	802	802	802	802	802	802	802	802	802
004	113	113	113	113	113	113	113	113	113	113	113	113	113	113	113	113	113	113	113	113	113
005	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106
006	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106
007	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106
008	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109
009	807	807	807	807	807	807	807	807	807	807	807	807	807	807	807	807	807	807	807	807	807
010	807	807	807	807	807	807	807	807	807	807	807	807	807	807	807	807	807	807	807	807	807
011	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109
012	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109
013	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111
014	808	808	808	808	808	808	808	808	808	808	808	808	808	808	808	808	808	808	808	808	808
015	808	808	808	808	808	808	808	808	808	808	808	808	808	808	808	808	808	808	808	808	808
016	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111
017	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111
018	809	809	809	809	809	809	809	809	809	809	809	809	809	809	809	809	809	809	809	809	809
019	114	114	114	114	114	114	114	114	114	114	114	114	114	114	114	114	114	114	114	114	114
020	115	115	115	115	115	115	115	115	115	115	115	115	115	115	115	115	115	115	115	115	115
021	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116
022	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118
023	810	810	810	810	810	810	810	810	810	810	810	810	810	810	810	810	810	810	810	810	810
024	810	810	810	810	810	810	810	810	810	810	810	810	810	810	810	810	810	810	810	810	810
025	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118
026	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118
027	119	119	119	119	119	119	119	119	119	119	119	119	119	119	119	119	119	119	119	119	119
028	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120
029	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121
030	812	812	812	812	812	812	812	812	812	812	812	812	812	812	812	812	812	812	812	812	812
031	812	812	812	812	812	812	812	812	812	812	812	812	812	812	812	812	812	812	812	812	812
032	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121
033	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121
034	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123
035	813	813	813	813	813	813	813	813	813	813	813	813	813	813	813	813	813	813	813	813	813
036	813	813	813	813	813	813	813	813	813	813	813	813	813	813	813	813	813	813	813	813	813
037	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123
038	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123
039	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124
040	814	814	814	814	814	814	814	814	814	814	814	814	814	814	814	814	814	814	814	814	814
041	814	814	814	814	814	814	814	814	814	814	814	814	814	814	814	814	814	814	814	814	814
042	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124
043	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124
044	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126
045	815	815	815	815	815	815	815	815	815	815	815	815	815	815	815	815	815	815	815	815	815
046	815	815	815	815	815	815	815	815	815	815	815	815	815	815	815	815	815	815	815	815	815
047	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126
048	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126
049	049	049	049	049	049	049	049	049	049	049	049	049	049	049	049	049	049	049	049	049	049

Tabla 12: Segunda parte de Matriz de Transición

	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43
	W	X	Y	Z	ESP	;	:	,	.	+	-	*	/	=	%	#	(	)	[	]	0..9	otro
000	034	039	804	044	197	198	810	810	810	810	810	810	810	810	198	810	049	810	810	810	810	803
001	801	801	801	801	801	801	801	801	801	801	801	801	801	801	801	801	801	801	801	801	002	801
002	107	107	107	107	107	107	107	107	107	107	107	107	107	107	107	107	107	107	107	107	002	107
003	802	802	802	802	802	802	802	802	802	802	802	802	802	802	802	802	802	802	802	802	004	802
004	113	113	113	113	113	113	113	113	113	113	113	113	113	113	113	113	113	113	113	113	004	113
005	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	006	106
006	106	106	106	106	106	106	106	106	007	106	106	106	106	106	106	106	106	106	106	106	006	106
007	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	007	106
008	109	109	109	109	109	109	109	109	009	010	010	109	109	109	109	109	109	109	109	109	011	109
009	807	807	807	807	807	807	807	807	807	807	807	807	807	807	807	807	807	807	807	807	012	807
010	807	807	807	807	807	807	807	807	009	807	807	807	807	807	807	807	807	807	807	807	011	807
011	109	109	109	109	109	109	109	109	012	109	109	109	109	109	109	109	109	109	109	109	011	109
012	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	109	012	109
013	111	111	111	111	111	111	111	111	014	015	015	111	111	111	111	111	111	111	111	111	016	111
014	808	808	808	808	808	808	808	808	808	808	808	808	808	808	808	808	808	808	808	808	017	808
015	808	808	808	808	808	808	808	808	014	808	808	808	808	808	808	808	808	808	808	808	016	808
016	111	111	111	111	111	111	111	111	017	111	111	111	111	111	111	111	111	111	111	111	016	111
017	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	111	017	111
018	809	809	809	809	809	809	809	809	809	809	809	809	809	809	809	809	809	809	809	809	019	809
019	114	114	114	114	114	114	114	114	114	114	114	114	114	114	114	114	114	114	114	114	019	114
020	115	115	115	115	115	115	115	115	115	115	115	115	115	115	115	115	115	115	115	115	020	115
021	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	021	116
022	118	118	118	118	118	118	118	118	023	024	024	118	118	118	118	118	118	118	118	118	025	118
023	810	810	810	810	810	810	810	810	810	810	810	810	810	810	810	810	810	810	810	810	026	810
024	810	810	810	810	810	810	810	810	023	810	810	810	810	810	810	810	810	810	810	810	025	810
025	118	118	118	118	118	118	118	118	026	118	118	118	118	118	118	118	118	118	118	118	025	118
026	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	026	118
027	119	119	119	119	119	119	119	119	119	119	119	119	119	119	119	119	119	119	119	119	027	119
028	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	028	120
029	121	121	121	121	121	121	121	121	030	031	031	121	121	121	121	121	121	121	121	121	032	121
030	812	812	812	812	812	812	812	812	812	812	812	812	812	812	812	812	812	812	812	812	033	812
031	812	812	812	812	812	812	812	812	030	812	812	812	812	812	812	812	812	812	812	812	032	812
032	121	121	121	121	121	121	121	121	033	121	121	121	121	121	121	121	121	121	121	121	032	121
033	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	121	033	121
034	123	123	123	123	123	123	123	123	035	036	036	123	123	123	123	123	123	123	123	123	037	123
035	813	813	813	813	813	813	813	813	813	813	813	813	813	813	813	813	813	813	813	813	038	813
036	813	813	813	813	813	813	813	813	035	813	813	813	813	813	813	813	813	813	813	813	037	813
037	123	123	123	123	123	123	123	123	038	123	123	123	123	123	123	123	123	123	123	123	037	123
038	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	038	123
039	124	124	124	124	124	124	124	124	040	041	041	124	124	124	124	124	124	124	124	124	042	124
040	814	814	814	814	814	814	814	814	814	814	814	814	814	814	814	814	814	814	814	814	043	814
041	814	814	814	814	814	814	814	814	040	814	814	814	814	814	814	814	814	814	814	814	042	814
042	124	124	124	124	124	124	124	124	043	124	124	124	124	124	124	124	124	124	124	124	042	124
043	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	124	043	124
044	126	126	126	126	126	126	126	126	045	046	046	126	126	126	126	126	126	126	126	126	047	126
045	815	815	815	815	815	815	815	815	815	815	815	815	815	815	815	815	815	815	815	815	048	815
046	815	815	815	815	815	815	815	815	045	815	815	815	815	815	815	815	815	815	815	815	047	815
047	126	126	126	126	126	126	126	126	048	126	126	126	126	126	126	126	126	126	126	126	047	126
048	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	048	126
049	049	049	049	049	049	049	049	049	049	049	049	049	049	049	049	049	950	199	049	049	049	049

Diagramas de Transición

Figura 12: Velocidad de alimentación

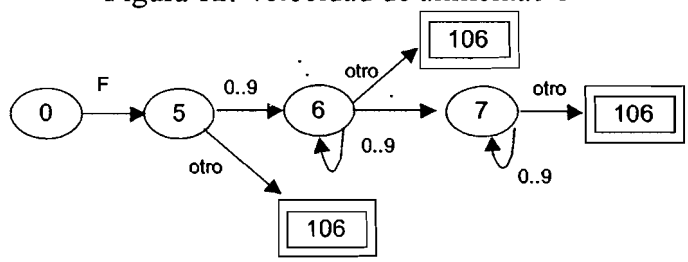


Figura 13: Funciones Preparatorias

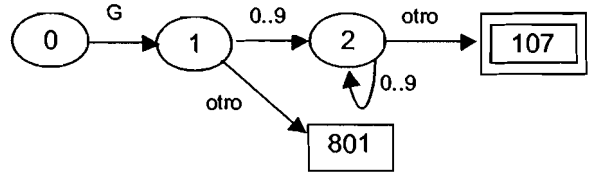


Figura 14: Variable de Posicionamiento I

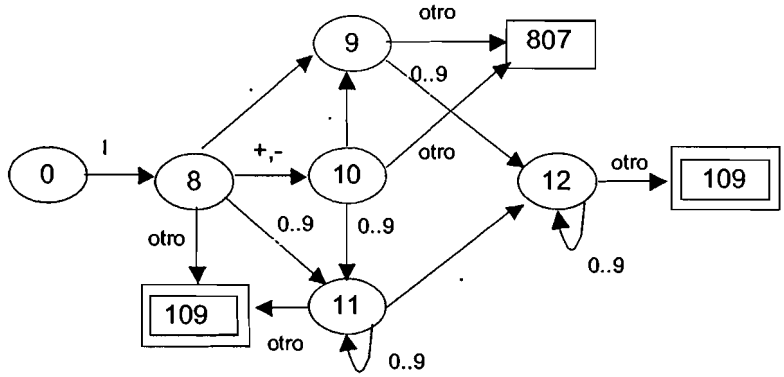


Figura 15: Variable de Posicionamiento K

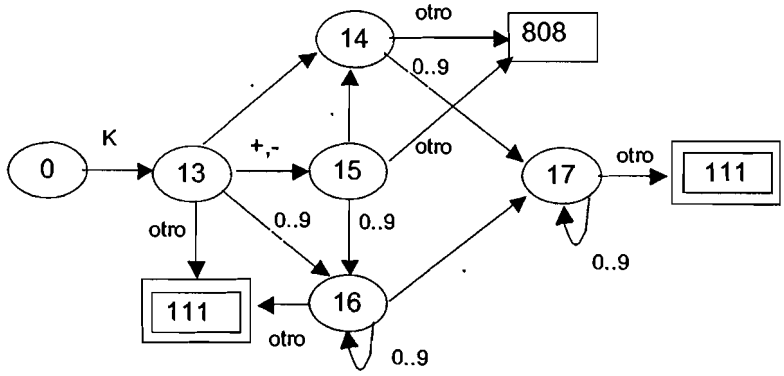


Figura 16: Funciones Miscelaneas

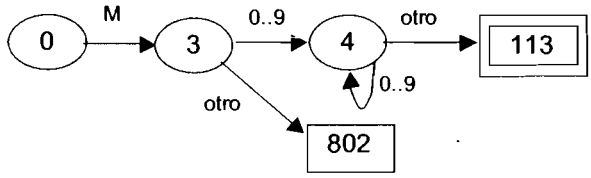


Figura 17: Número de Bloque

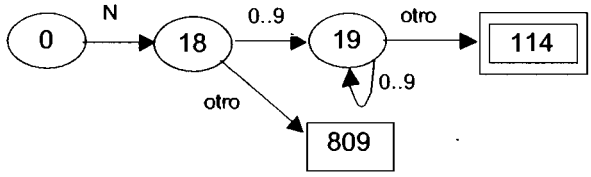


Figura 18: Nombre de Programa

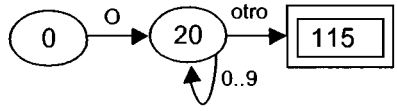


Figura 19: Parámetro P

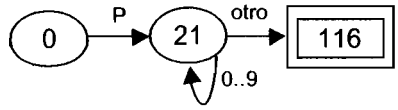


Figura 20: Velocidad del Husillo

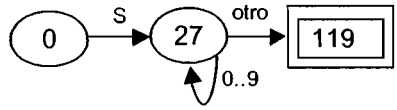


Figura 21: Número de Herramnienta

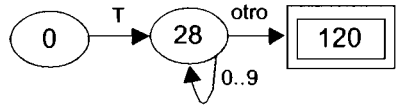


Figura 22: Variable de Posicionamiento U

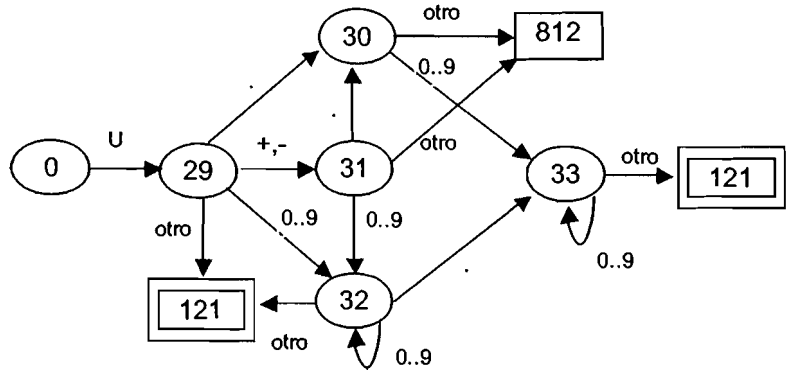


Figura 23: Variable de Posicionamiento W

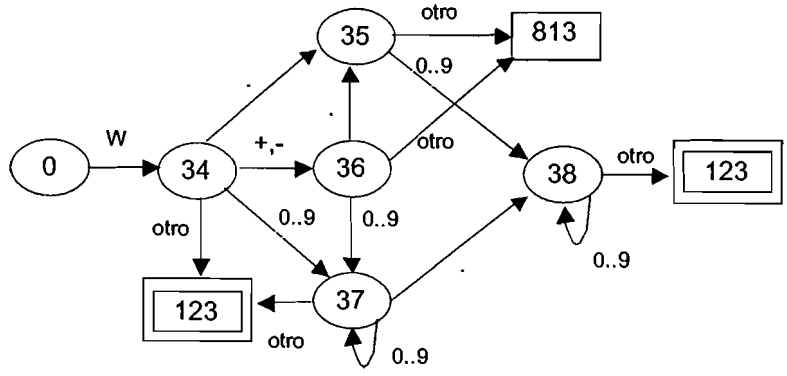


Figura 24: Variable de Posicionamiento X

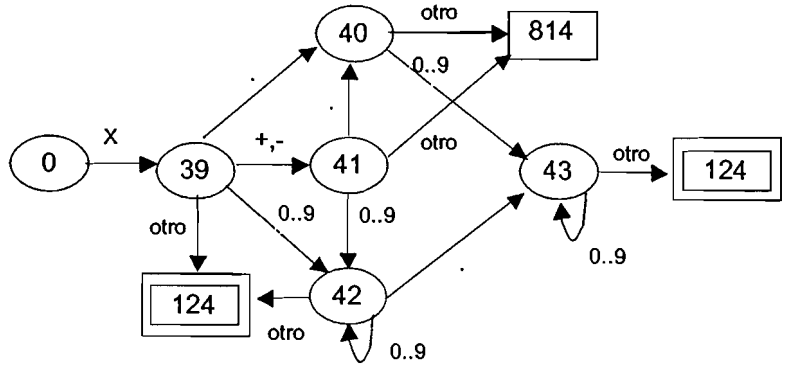


Figura 25: Variable de Posicionamiento Z

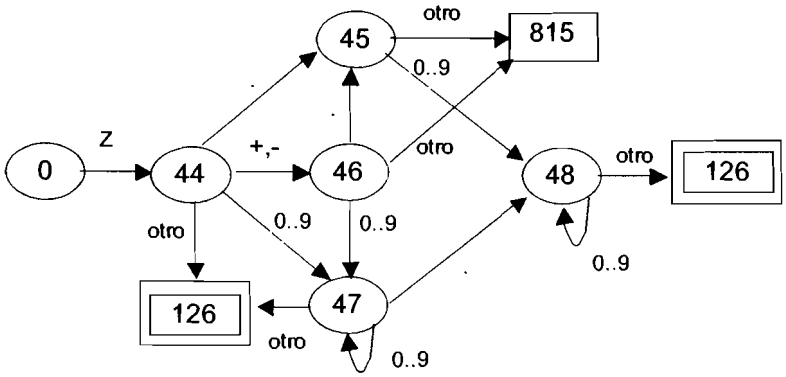


Figura 26: Comentario de una línea

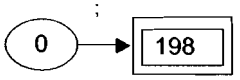


Figura 27: Comentario de una o más líneas

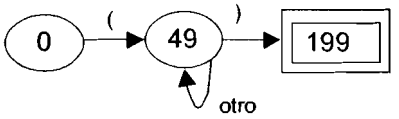
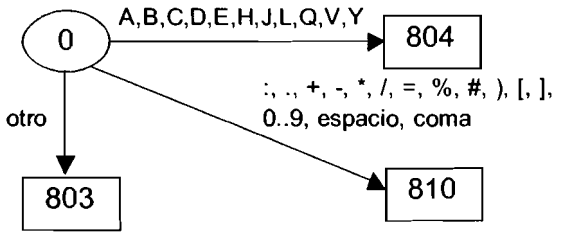


Figura 28: Cualquier otro valor



Programa: Compile.cpp

```
//
// Universidad Autónoma de Querétaro
//
// Descripción:              Compilador de código G
//
// Autor:                    Aurora Femat Díaz
//
// Ultima modificación:      4 de Diciembre de 2004
//---------------------------------------------------------------------------

#include <vcl.h>
#include <stdlib.h>

#pragma hdrstop

#include "Compile.h"
#include "Main.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TCompilerForm *CompilerForm;
//---------------------------------------------------------------------------
__fastcall TCompilerForm::TCompilerForm(TComponent* Owner)
      : TForm(Owner)
{
  int i,j,k;
  int row;
  String S,S2;

  // Por el momento para configurar el idioma del compilador
  _language = 2;
  CDir="c:\\Documents and Settings\\Aurora\\Mis Documentos\\Mi tesis";

  //*************************************************************************
  //  Carga tabla de FUNCIONES PREPARATORIAS
  //  archivo compuesto por nombre, numero, desc. en inglés y desc. en español
  //*************************************************************************

  TStringList* linesPF = new TStringList;
  try {
    linesPF->LoadFromFile(CDir+"\\Database\\PreparatoryFunctions.txt");
  }
  catch (...) {
    MessageBox(Handle,"Error al intentar crear la tabla de las Funciones Preparatorias",
"Message", 0);
    return;
  }
  for (i=0;i<linesPF->Count && i<CPreFunc;i++)
```

```
  linesParam->LoadFromFile(CDir+"\\Database\\Parameters.txt");
}
catch (...) {
  MessageBox(Handle,"Error Creating Parameters Table", "Message", 0);
  return;
}
for (i=0;i<linesParam->Count && i<CParamInOut;i++)
{
  //
  //  Obten la linea i del archivo
  //
  S=linesParam->Strings[i];
  //
  // Número de error
  //
  Funciones[i].Function  = S.SubString(1, 2).ToInt();
  Funciones[i].DataIn.X  = (S.SubString( 4, 1) == "1") ? true : false;
  Funciones[i].DataIn.Z  = (S.SubString( 6, 1) == "1") ? true : false;
  Funciones[i].DataIn.U  = (S.SubString( 8, 1) == "1") ? true : false;
  Funciones[i].DataIn.W  = (S.SubString(10, 1) == "1") ? true : false;
  Funciones[i].DataIn.I  = (S.SubString(12, 1) == "1") ? true : false;
  Funciones[i].DataIn.K  = (S.SubString(14, 1) == "1") ? true : false;
  Funciones[i].DataIn.R  = (S.SubString(16, 1) == "1") ? true : false;
  Funciones[i].DataIn.P  = (S.SubString(18, 1) == "1") ? true : false;

  Funciones[i].DataOut.U = (S.SubString(20, 1) == "1") ? true : false;
  Funciones[i].DataOut.W = (S.SubString(22, 1) == "1") ? true : false;
  Funciones[i].DataOut.R = (S.SubString(24, 1) == "1") ? true : false;
}
delete linesParam;

//*************************************************************************
//  Carga tabla de Offset de la herramienta
//  archivo compuesto por coordenada en x,y,z
//*************************************************************************

TStringList* linesOST = new TStringList;
try {
  linesOST->LoadFromFile(CDir+"\\Database\\OffsetsTool.txt");
}
catch (...) {
  MessageBox(Handle,"Error Creating Tool Offsets Table", "Message", 0);
  return;
}
for (i=0;i<linesOST->Count && i<COffsetTool;i++)
{
  //
  //  Obten la linea i del archivo
  //
```

```
}
catch (...) {
  MessageBox(Handle,"Error Creating Miscellaneous Functions Table", "Message", 0);
  return;
}
for (i=0;i<linesMF->Count && i<CMisFunc;i++)
{
  //
  //  Obten la linea i del archivo
  //
  S=linesMF->Strings[i];
  //
  // Grupo al que pertenece
  //
  int pos = S.Pos(";");
  S2 = S.SubString(1, pos-1);
  MMisFunc[i].Group=S2.ToInt();
  S.Delete(1,pos);
  //
  // Instante para Aplicacion
  //
  pos = S.Pos(";");
  S2 = S.SubString(1, pos-1);
  MMisFunc[i].AplicMoment=S2.ToInt();
  S.Delete(1,pos);
  //
  // Nombre de la funcion miscelanea
  //
  pos = S.Pos(";");
  S2 = S.SubString(1, pos-1);
  MMisFunc[i].NameFunc=S2;
  S.Delete(1,pos);
  //
  // Número de la funcion miscelanea
  //
  pos = S.Pos(";");
  S2 = S.SubString(1, pos-1);
  MMisFunc[i].NumbFunc=S2.ToInt();
  S.Delete(1,pos);
  //
  // Descripción en inglés de la funcion miscelanea
  //
  pos = S.Pos(";");
  S2 = S.SubString(1, pos-1);
  MMisFunc[i].DesEFunc=S2;
  S.Delete(1,pos);
  //
  // Descripción en español de la funcion miscelanea
  //
```

```
  MMisFunc[i].DesSFunc=S;
  MMisFunc[i].Actived=false;
}
delete linesMF;

//*****************************************************************************
//  Carga tabla de Descripcion de Errores
//  archivo compuesto por numero, desc. en inglés y desc. en español
//*****************************************************************************

TStringList* linesEM = new TStringList;
try {
  linesEM->LoadFromFile(CDir+"\\Database\\ErrorMessages.txt");
}
catch (...) {
  MessageBox(Handle,"Error Creating Error Messages Table", "Message", 0);
  return;
for (i=0;i<linesEM->Count && i<CErrMess;i++)
{
  //
  //  Obten la linea i del archivo
  //
  S=linesEM->Strings[i];
  //
  // Número de error
  //
  int pos = S.Pos(";");
  MErrMess[i].NumbErrM=S2;
  S.Delete(1,pos);
  //
  // Descripción en inglés del error
  //
  pos = S.Pos(";");
  S2 = S.SubString(1, pos-1);
  MErrMess[i].DesEErrM=S2;
  S.Delete(1,pos);
  //
  // Descripción en español del error
  //
  MErrMess[i].DesSErrM=S;
}
delete linesEM;

//*****************************************************************************
//  Carga datos para MATRIZ DE TRANSICION
//*****************************************************************************
TStringList* linesMT = new TStringList;
try {
  linesMT->LoadFromFile(CDir+"\\Database\\MatTrans.txt");
```

```
}
catch (...) {
  MessageBox(Handle,"Error Creating Transition Table", "Message", 0);
  return;
}
//
// linesMT->Count debe ser el numero de estados
//
for (i=0;i<linesMT->Count && i<CRowsMT;i++)
{
  //
  //  Obten la linea i del archivo
  //
  S=linesMT->Strings[i];
  for (j=0;j<CColuMT-1;j++)
  {
    //
    //  Encuentra dato i,j-ésimo
    //
    int pos = S.Pos(";");
    S2 = S.SubString(1, pos-1);
    S.Delete(1,pos);
    //
    //  Escribe dato encontrado a la matriz de transición
    //
    MatTrans[i][j]= S2.ToInt();
  }
  MatTrans[i][CColuMT-1]=S.ToInt();
}
delete linesMT;
//*****************************************************************************
//  termina carga de matriz de transicion
//*****************************************************************************

//*****************************************************************************
//  Carga datos para MATRIZ DE PARSEO
//*****************************************************************************
TStringList* linesMP = new TStringList;
try {
  linesMP->LoadFromFile(CDir+"\\Database\\MatPars.txt");
}
catch (...) {
  MessageBox(Handle,"Error Creating Parser Table", "Message", 0);
  return;
}
//
// RENGLONES
//
for (i=0;i<linesMP->Count && i<CRowsMP;i++)
```

```
{
  //
  //  Obten la linea i del archivo
  //
  S=linesMP->Strings[i];
  for (j=0;j<CColuMP;j++)
  {
    int pos;
    //
    //  Encuentra dato i,j-ésimo para las profundidades 0,1,2,3
    //
    for (k=0;k<4;k++)
    {
      pos = S.Pos(",");
      S2 = S.SubString(1, pos-1);
      S.Delete(1,pos);
      MatParseo[i][j][k]= S2.ToInt();
    }
  }
}
delete linesMP;
//*****************************************************************************
//  termina carga de matriz de PARSEO
//*****************************************************************************

//*****************************************************************************
//  Carga tabla de Limites de Variables:
//  Nombre de variable,P o N parámetro de posicionamiento o no,
//  D o E para decimal o entero,Número de token,
//  lim.inf. para variable en mm., lim.sup. para variable en mm
//*****************************************************************************

TStringList* linesLIM = new TStringList;
try {
  linesLIM->LoadFromFile(CDir+"\\Database\\Limits.txt");
}
catch (...) {
  MessageBox(Handle,"Error Creating Variable Limits Table", "Message", 0);
  return;
}
//
// linesLIM->Count debe ser el numero de variables para movimiento
//
for (i=0;i<linesLIM->Count && i<CFunVarLim;i++)
{
  S=linesLIM->Strings[i];
  //
  // Nombre de variable
  //
```

```
  int pos = S.Pos(";");
  S2 = S.SubString(1, pos-1);
  S.Delete(1,pos);
  FunVarData[i].Name=S2;
  //
  // P param. posicionamiento; N, no
  //
  pos = S.Pos(";");
  S2 = S.SubString(1, pos-1);
  S.Delete(1,pos);
  FunVarData[i].PorN=S2;
  //
  // E p/enteros, D p/decimales
  //
  pos = S.Pos(";");
  S2 = S.SubString(1, pos-1);
  S.Delete(1,pos);
  FunVarData[i].IorF=S2;
  //
  // Num. de Token
  //
  pos = S.Pos(";");
  S2 = S.SubString(1, pos-1);
  S.Delete(1,pos);
  FunVarData[i].NumToken=S2.ToDouble();
  //
  // lim inf de var en mm
  //
  pos = S.Pos(";");
  S2 = S.SubString(1, pos-1);
  S.Delete(1,pos);
  FunVarData[i].lim_inf=S2.ToDouble();
  //
  // lim sup de var en mm
  //
  FunVarData[i].lim_sup=S.ToDouble();
}
delete linesLIM;
//***************************************************************************
//  termina carga de limites de variables
//***************************************************************************

//***************************************************************************
//  Carga tabla de Parametro de Entrada y Salida
//  arch. compuesto por número de función y parametros X;Z;U;W;I;K;R;U;W;R;
//***************************************************************************

TStringList* linesParam = new TStringList;
try {
```

```
  linesParam->LoadFromFile(CDir+"\\Database\\Parameters.txt");
}
catch (...) {
  MessageBox(Handle,"Error Creating Parameters Table", "Message", 0);
  return;
}
for (i=0;i<linesParam->Count && i<CParamInOut;i++)
{
  //
  //  Obten la linea i del archivo
  //
  S=linesParam->Strings[i];
  //
  // Número de error
  //
  Funciones[i].Function  = S.SubString(1, 2).ToInt();
  Funciones[i].DataIn.X  = (S.SubString( 4, 1) == "1") ? true : false;
  Funciones[i].DataIn.Z  = (S.SubString( 6, 1) == "1") ? true : false;
  Funciones[i].DataIn.U  = (S.SubString( 8, 1) == "1") ? true : false;
  Funciones[i].DataIn.W  = (S.SubString(10, 1) == "1") ? true : false;
  Funciones[i].DataIn.I  = (S.SubString(12, 1) == "1") ? true : false;
  Funciones[i].DataIn.K  = (S.SubString(14, 1) == "1") ? true : false;
  Funciones[i].DataIn.R  = (S.SubString(16, 1) == "1") ? true : false;
  Funciones[i].DataIn.P  = (S.SubString(18, 1) == "1") ? true : false;

  Funciones[i].DataOut.U = (S.SubString(20, 1) == "1") ? true : false;
  Funciones[i].DataOut.W = (S.SubString(22, 1) == "1") ? true : false;
  Funciones[i].DataOut.R = (S.SubString(24, 1) == "1") ? true : false;
}
delete linesParam;

//***************************************************************************
//  Carga tabla de Offset de la herramienta
//  archivo compuesto por coordenada en x,y,z
//***************************************************************************

TStringList* linesOST = new TStringList;
try {
  linesOST->LoadFromFile(CDir+"\\Database\\OffsetsTool.txt");
}
catch (...) {
  MessageBox(Handle,"Error Creating Tool Offsets Table", "Message", 0);
  return;
}
for (i=0;i<linesOST->Count && i<COffsetTool;i++)
{
  //
  //  Obten la linea i del archivo
  //
```

```
  S=linesOST->Strings[i];
  //
  // Valor en X
  //
  int pos = S.Pos(";");
  S2 = S.SubString(1, pos-1);
  OffsetTool[i].X=S2.ToDouble();
  S.Delete(1,pos);
  //
  // Valor en Y
  //
  pos = S.Pos(";");
  S2 = S.SubString(1, pos-1);
  OffsetTool[i].Y=S2.ToDouble();
  S.Delete(1,pos);
  //
  // Valor en Z
  //
  OffsetTool[i].Z=S.ToDouble();
}
delete linesOST;

//*****************************************************************************
//  Carga tabla de Offset de trabajo
//  archivo compuesto por coordenada en x,y,z
//*****************************************************************************

TStringList* linesOSW = new TStringList;
try {
  linesOSW->LoadFromFile(CDir+"\\Database\\OffsetsWork.txt");
}
catch (...) {
  MessageBox(Handle,"Error Creating Work Offsets Table", "Message", 0);
  return;
}
for (i=0;i<linesOSW->Count && i<COffsetWork;i++)
{
  //
  //  Obten la linea i del archivo
  //
  S=linesOSW->Strings[i];
  //
  // Valor en X
  //
  int pos = S.Pos(";");
  S2 = S.SubString(1, pos-1);
  OffsetWork[i].X=S2.ToDouble();
  S.Delete(1,pos);
  //
```

```
    // Valor en Y
    //
    pos = S.Pos(";");
    S2 = S.SubString(1, pos-1);
    OffsetWork[i].Y=S2.ToDouble();
    S.Delete(1,pos);
    //
    // Valor en Z
    //
    OffsetWork[i].Z=S.ToDouble();
  }
  delete linesOSW;

  //*************************************************************************
  //  termina carga de definicion de parametros para funciones
  //*************************************************************************


//=========================================================================================
======
  //
  // Tablas para visializacion en el desarrollo
  //
  StrGridPreFunc->RowCount=2;
  StrGridMisFunc->RowCount=2;
  StrGridPreFunc->Cells[2][0]="Num";
  StrGridPreFunc->Cells[3][0]="PreFunc";
  StrGridPreFunc->Cells[4][0]="EngDesc";
  StrGridPreFunc->Cells[5][0]="SpaDesc";

  StrGridMisFunc->Cells[1][0]="Gr";
  StrGridMisFunc->Cells[2][0]="AplMoment";
  StrGridMisFunc->Cells[3][0]="ElapsedTime";
  StrGridMisFunc->Cells[4][0]="Num";
  StrGridMisFunc->Cells[5][0]="MisFunc";
  StrGridMisFunc->Cells[6][0]="EngDesc";
  StrGridMisFunc->Cells[7][0]="SpaDesc";
  for (i=1;i<=8;i++)
  {
    StrGridPreFunc->Cells[i][1]="";
    StrGridMisFunc->Cells[i][1]="";
  }
  row = StrGridPreFunc->RowCount;
  for (i=0;i<CPreFunc;i++)
  {
    StrGridPreFunc->Cells[0][i+row-1]=i+row-1;
    StrGridPreFunc->Cells[1][i+row-1]=MPreFunc[i].Group;
    StrGridPreFunc->Cells[2][i+row-1]=MPreFunc[i].NumbFunc;
    StrGridPreFunc->Cells[3][i+row-1]=MPreFunc[i].NameFunc;
```

```
    StrGridPreFunc->Cells[4][i+row-1]=MPreFunc[i].DesEFunc;
    StrGridPreFunc->Cells[5][i+row-1]=MPreFunc[i].DesSFunc;
    StrGridPreFunc->RowCount++;
  }
  row = StrGridMisFunc->RowCount;
  for (i=0;i<CMisFunc;i++)
  {
    StrGridMisFunc->Cells[0][i+row-1]=i+row-1;
    StrGridMisFunc->Cells[1][i+row-1]=MMisFunc[i].Group;
    StrGridMisFunc->Cells[2][i+row-1]=MMisFunc[i].AplicMoment;
    StrGridMisFunc->Cells[3][i+row-1]=MMisFunc[i].NumbFunc;
    StrGridMisFunc->Cells[4][i+row-1]=MMisFunc[i].NameFunc;
    StrGridMisFunc->Cells[5][i+row-1]=MMisFunc[i].DesEFunc;
    StrGridMisFunc->Cells[6][i+row-1]=MMisFunc[i].DesSFunc;
    StrGridMisFunc->RowCount++;
  }
  row = StrGridErrMess->RowCount;
  for (i=0;i<CErrMess;i++)
  {
    StrGridErrMess->Cells[0][i+row-1]=i+row-1;
    StrGridErrMess->Cells[1][i+row-1]=MErrMess[i].NumbErrM;
    StrGridErrMess->Cells[2][i+row-1]=MErrMess[i].DesEErrM;
    StrGridErrMess->Cells[3][i+row-1]=MErrMess[i].DesSErrM;
    StrGridErrMess->RowCount++;
  }
  row = StrGridVarLimits->RowCount;
  for (i=0;i<CFunVarLim;i++)
  {
    StrGridVarLimits->Cells[0][i+row-1]=i+row-1;
    StrGridVarLimits->Cells[1][i+row-1]=FunVarData[i].Name;
    StrGridVarLimits->Cells[2][i+row-1]=FunVarData[i].IorF;
    StrGridVarLimits->Cells[3][i+row-1]=FunVarData[i].NumToken;
    StrGridVarLimits->Cells[4][i+row-1]=FunVarData[i].lim_inf;
    StrGridVarLimits->Cells[5][i+row-1]=FunVarData[i].lim_sup;
    StrGridVarLimits->RowCount++;
  }
  //
  //

//=============================================================================
=====
}
//---------------------------------------------------------------------------

//
// Inicializa los valores de los grupos G y M por omisión.
// Inicializa los valores de las variables por programa,
// Mientras no se encuentre con el final de archivo que está compilando
// para cada línea del archivo:
```

```
//    solicita el siguiente token, si el número de token proporcionado es:
//    mayor a 800 hay un error de léxico,
//    107 función G, 113 función M, en estos dos últimos casos verifica que
//    dicha función es válida;
//    si no ha habido error, llama al parseador con línea y columna de parseo
//    llama función para verificar cambio de línea y ahí se
//    genere código si hay pendiente a ser generado
//    si ha habido algún error se agrega a lista de errores
//
int TCompilerForm::OnCompile(AnsiString _NameFile)
{
  bool       swErr;
  int        i,_line,_column,_lant,_cant;
  int        _error,_cnt_errors,_cnt_lines;
  int        _lParseo,_cParseo;
  AnsiString  MsgErr,idErr;
  typeNumToken NumToken;

  typeToken  StrToken,StrTokenAnt,StrTokenErr;
  AnsiString MsgError;
  PreparatoryFunctionRecord   *RPreFunc;
  MiscellaneousFunctionRecord  *RMisFunc;

  _error      = 0;
  _line       = 0;
  _lant       = 0;
  _column     = 1;
  _cant       = 1;
  _lParseo    = 0;
  _cParseo    = 0;
  _cnt_errors = 0;
  UPosition   = 0;
  WPosition   = 0;
  XPosition   = 0;
  ZPosition   = 0;
  TotBlks     = 0;
  topNLine    = 0;
  headTokens  = NULL;
  headErrors  = NULL;
  StrTokenErr = "";
  _cnt_lines  = CopyMemoEditor->Lines->Count;
  //
  // Actualiza los textos de la forma de resultados
  //
  CompileResultForm->Show();
  CompileResultForm->ModalResult=true;
  CompileResultForm->LblTotalLines->Caption = _cnt_lines;
  if (_NameFile.Length()>25)
```

```
    CompileResultForm->LblFileName->Caption   =
"..."+_NameFile.SubString(_NameFile.Length()-24,25);
  else
    CompileResultForm->LblFileName->Caption   = _NameFile;
  CompileResultForm->LblResult->Caption     = "Compiling...";
  CompileResultForm->LblErrors->Caption     = "0";
  CompileResultForm->LblCurrentLine->Caption = "0";
  //
  CompileResultForm->Refresh();
  //
  // Borra los bloques generados si es una segunda o posterior compilacion
  //
  MemoBlocks->Lines->Clear();

  for (i=0;i<CGGroups;i++)
    ActiveGValues[i] = -1;
  for (i=0;i<CMGroups;i++)
    ActiveMValues[i] = -1;
  //
  // Valores por omisión para cada grupo
  //
  ActiveGValues[1]  =  0;  // G de movimiento,          G00
  ActiveGValues[6]  =  21; // Unidades de entrada,      G21 (mm)
  ActiveGValues[12] =  54; // Offset de trabajo         G54 (0)

  //
  // Inicializa Valiables de bloque y de programa
  //
  IniVariablesPrg();
  IniVariablesBlk();
  //
  // Para cada una de las líneas del programa
  //
  while (_line<_cnt_lines)
  {
    //
    // Solicita un token
    //
    StrTokenAnt = StrToken;
    NumToken=Token(&_line, &_column, &_lant,&_cant, &StrToken, _cnt_lines);
    //
    // Si el token identificado no es comentario
    //
    if (!(NumToken > 197 && NumToken <200) && PrgRec.M30)
    {
      _error = 53;
      InsErrorWarning(_error,"",&_cnt_errors,_lant,_cant);
      PrgRec.M30 = false;
      _error = 0;
```

```
}
//
// Ocurrio un error al identificar el token con su parámetro
//
if (NumToken > 800)
{
  _error = NumToken-800;
  StrToken=CopyMemoEditor->Lines->Strings[_line].SubString(_cant,_column-_cant+1);
  StrTokenErr = StrToken;
  if (_line<_cnt_lines)
  {
    _column=1;
    _line++;
  }
}
else
{
  //
  //   token 198 es ; salta el resto de la línea
  //
  if (NumToken>0 && NumToken<198)  // pendiente cuando sea comentario 198
  {
    if (NumToken == 107)
    {
      //
      // busca descripcion para funcion prep.
      //
      _error = !PreFuncRecord(StrToken,&RPreFunc);
      if (_error)
         _error = 11;
      else
         NumToken = (*RPreFunc).NumbFunc;
    }
    if (NumToken == 113)
    {
      //
      // busca descripcion para funcion miscelanea
      //
      _error = !MisFuncRecord(StrToken,&RMisFunc);
      if (_error)
         _error = 12;
      else
         NumToken = (*RMisFunc).NumbFunc;
    }
    if (!_error)
      _error=Parser(NumToken,StrToken,&_lParseo, &_cParseo,&RPreFunc,&RMisFunc);
    if (!_error)
    {
     InsToken(NumToken,StrToken,_lant);
```

```
        swErr = false;
        CompileResultForm->LblErrors->Caption = _cnt_errors;
      }
      else
      {
        swErr = true;
        StrTokenErr = StrToken;
      }
      //
      // Hubo cambio de linea???
      //
      ChkChgLine(_line, _lant, &_error);
      if (!swErr && _error)
      {
        StrTokenErr = "";
      }
    } // if (NumToken>0 && NumToken<198)
  } // else (NumToken > 800)
  if (_error )
  {
    if (StrTokenErr.Length() < 7)
      idErr = StrTokenErr;
    else
      idErr = StrTokenErr.SubString(1,4) + "...";
    InsErrorWarning(_error,idErr,&_cnt_errors,_lant,_cant);
    _error = 0;
    swErr = false;
    CompileResultForm->LblErrors->Caption   = IntToStr(_cnt_errors);
  } // if (_error || _warning )
  CompileResultForm->LblCurrentLine->Caption = IntToStr(_lant+1);
  CompileResultForm->Update();
} // while ((*_line)<_cnt_lines)
if (!PrgRec.M30)
{
  _error = 52;
  InsErrorWarning(_error,"",&_cnt_errors,_lant,_cant);
}
FreeTokens();
FreeNLine();
CompileResultForm->LblErrors->Caption   = IntToStr(_cnt_errors);
CompileResultForm->LblCurrentLine->Caption = "0";
if (_cnt_errors > 0)
{
  FreeErrors();
  CompileResultForm->LblResult->Caption     = "Done: There are errors.";
}
else
  CompileResultForm->LblResult->Caption     = "Done:        Make.";
CompileResultForm->Visible=false;
```

```
  CompileResultForm->ShowModal();
  return _cnt_errors;
}
//---------------------------------------------------------------------------
void TCompilerForm::InsErrorWarning(int _error,AnsiString idErr,
                int *_cnt_errors,int _lant,int _cant)

{
  AnsiString msg;

  if (_error)
  {
    (*_cnt_errors)++;
    if (DescErrMess(_error,&msg))
      InsError("L" + IntToStr(_lant+1),"C" + IntToStr(_cant),
                  "E" + IntToStr(_error),msg,idErr);
    else
      InsError("L" + IntToStr(_lant+1),"C" + IntToStr(_cant),
                  "E" + IntToStr(_error),"",idErr);
  }
}
//
// De la matríz de parseo iniciando en línea y columna de parseo navega a
// través de la matriz hasta encontrar el token recibido y ejecuta su acción
// correspondiente.
// Mientras no se encuentre un estado aceptor (7) navega a través de la matriz
// como lo indique el estado de parseo. Este estado aceptor, llama a generación
// de código para que ejecute la acciones necesarias para generar el código o
// para las prepara para cuando haya un cambio de línea.
//
int TCompilerForm::Parser(typeNumToken _NumToken,typeToken StrToken,
              int *_lParseo, int *_cParseo,
              PreparatoryFunctionRecord **RG,
              MiscellaneousFunctionRecord **RM)
{
  int _error;
  int edoMatParseo;

  _error = 0;
  do {
    edoMatParseo = MatParseo[*_lParseo][*_cParseo][0];
    switch (edoMatParseo)
    {
      //
      // Si lo que hay en la matriz de Parseo en el renglon y columnas
      // actuales profundidad 1 es el número de token en proceso
      // salta a la columna indicada por la profundidad 2 y reliza la accion
      // de generacion de código indicada en la profundidad 3
      //
```

```
    case (1) :
    {
      if (MatParseo[*_lParseo][*_cParseo][1]==_NumToken)
      {
        if (MatParseo[*_lParseo][*_cParseo][3]!=0)
            _error=ActionsToGenCode(MatParseo[*_lParseo][*_cParseo][3],_NumToken,
                    StrToken,_lParseo,_cParseo,RG,RM);
        if (!_error)
          *_cParseo = MatParseo[*_lParseo][*_cParseo][2];
        else
          *_cParseo = 0;
      }
      else
        (*_cParseo)++;
      break;
    }
    //
    // Salta al renglon indicado en la profuncidad 1, sin edo aceptor
    //
    case (4) :
    {
      *_lParseo = MatParseo[*_lParseo][*_cParseo][1];
      *_cParseo = 0;
      break;
    }
    //
    // Ocurrio un error, el número de error es lo que está en la
    // profundidad 1
    //
    case (6) :
    {
      _error = MatParseo[*_lParseo][*_cParseo][1];
      break;
    }
    //
    // Estado aceptor
    //
    case (7) :
    {
      if (MatParseo[*_lParseo][*_cParseo][3]!=0)
          _error=ActionsToGenCode(MatParseo[*_lParseo][*_cParseo][3],
          _NumToken,StrToken,_lParseo,_cParseo,RG,RM);
      *_lParseo = 0;
      *_cParseo = 0;
      break;
    }
  }
} while ((edoMatParseo==1 || edoMatParseo==4) && !_error);
return _error;
```

```
}
//---------------------------------------------------------------------------

// Acciones para GENERACION DE CODIGO:
//
//    1: Encender grupo G al que pertenece con el valor de ese G en particular
//    2: Checar que el parámetro no haya sido definido antes en ese bloque,
//       si ya había sido definido, mandar error correspondiente a la pila
//       si no había sido definido, activarlo y definirlo
//    3: Para la función O verificar que sea la primera instruccion generada,
//       sino es así indicar error, por que el nombre de un programa solo
//       puede ser especificado al inicio del programa
//    4: Mover a la variable  en cuestion el parámetro antes de continuar
//    6: Encender G28
//    7: Encender G29
//    8: Encender G50
//    9: Encender G92
//   10: Activar el parámetro de skip function
//   11: Exact Stop check activado para ese bloque
//   12: Encender grupo M al que pertenece con el valor de ese M en particular
//
int TCompilerForm::ActionsToGenCode(int _op,typeNumToken _NumToken,
                typeToken StrToken,int *_lParseo, int *_cParseo,
                PreparatoryFunctionRecord **RG,
                MiscellaneousFunctionRecord **RM)
{
  int _error=0;

  switch (_op)
  {
    case 1:
    {
      //
      // Activar grupo al q pertenece la funcion con su respectivo parámetro
      //
      ActiveGValues[(**RG).Group]=(**RG).NumbFunc-CCtePreFunc;
      //
      // Si es G de movimiento revisar que no se haya activado otro G
      // con parámetros en la misma línea
      //
      if ((**RG).Group==1)
        if (!BlkRec.GMov.actived && !BlkRec.GWP.actived)
          BlkRec.GMov.actived = true;
          BlkRec.GMov.value   = (**RG).NumbFunc-CCtePreFunc;
        }
        else
          _error = 16;
      break;
    }
```

```
case 2:
{
  //
  // Checar que el parámetro no  haya sido activado antes en ese bloque,
  // si ya había sido activado, enviar error correspondiente a una pila;
  // si no había sido activado, activarlo y definirle su valor
  //
  _error = ChkActiveParameter(_NumToken,StrToken);
  break;
}
case 3:
{
  //
  // Asigna las variables O, F
  //
  _error = AssUniqueParameters(_NumToken,StrToken);
  break;
}
case 4:
{
  //
  // Asigna las funciones S, T y checa que no hayan sido
  // activadas antes en el mismo bloque
  //
  _error = ChkActiveParameter(_NumToken,StrToken);
  break;
}

//
// Si no ha sido activado ningún G que lleve parámetros:
//       Activar G04, G28, G29, G50 o G92 que no son de movimiento
//       pero llevan parámetros
//
case 5:
{
  if (!BlkRec.GWP.actived) // Movimiento activo SOLO para este bloque
  {
     BlkRec.GWP.actived = true;
     BlkRec.GWP.value   = 04;
  }
  else
     _error = 17;
  break;
}
case 6: {
  if (!BlkRec.GWP.actived)
  {
     BlkRec.GWP.actived = true;
     BlkRec.GWP.value   = 28;
```

```
  }
  else
     _error = 18;
  break;
}
case 7: {
  if (!BlkRec.GWP.actived)
  {
     BlkRec.GWP.actived = true;
     BlkRec.GWP.value   = 29;
  }
  else
     _error = 19;
  break;
}
case 8: {
  if (!BlkRec.GWP.actived)
  {
     BlkRec.GWP.actived = true;
     BlkRec.GWP.value   = 50;
  }
  else
     _error = 20;
  break;
}
case 9: {
  if (!BlkRec.GWP.actived)
  {
     BlkRec.GWP.actived = true;
     BlkRec.GWP.value   = 92;
  }
  else
     _error = 20;
  break;
}
//
// Skip Function activado, solo de ahi en adelante, mandar slash
//
case 10: {

  break;
}
//
// Exact stop check activado para el bloque en proceso, no es modal
//
case 11: {
  if (PrgRec.ExactSTP)
    PrgRec.ExactSTP = false;
  else
```

```
        PrgRec.ExactSTP = true;
      break;
    }
    case 12: {
    //
    // Activar grupo M al que pertenece la instrucción con el número de M
    // y encender dicha función
    //
      ActiveMValues[(**RM).Group] = _NumToken-CCteMisFunc; // Funciones M
      (**RM).Actived            = true;
      break;
    }
    case 13: {
      if (BlkRec.GWP.actived)
        _error = 32;
      else
      {
        //
        // Activar grupo al q pertenece la funcion con el número de la función
        //
        ActiveGValues[(**RG).Group]=(**RG).NumbFunc-CCtePreFunc;
        BlkRec.GWP.actived = true;
        BlkRec.GWP.value   = (**RG).NumbFunc-CCtePreFunc;
        }
      break;
    }
    case 14:
    {
      //
      // Activar grupo al q pertenece la funcion con el número de la función
      // y encender parámetro para generar bloque
      //
      ActiveGValues[(**RG).Group]=(**RG).NumbFunc-CCtePreFunc;
      (**RG).Actived = true;
      break;
    }
    case 15:
    {
      //
      // Instrucción para terminar el programa
      //
      PrgRec.M30 = true;
      break;
    }

  }
  return _error;
}
//---------------------------------------------------------------------------
```

```
int TCompilerForm::ChkNumber(typeNumToken NumToken,typeToken StrToken,
                    float *_valor, int *_subind)
{
  bool        swEnc;
  int         _error,lon,i,NHerr,NOS;
  AnsiString  partToken;
  FunVarLimType funvarLimits;

  _error    = 0;
  lon       = StrToken.Length()-1;
  partToken  = StrToken.SubString(2,lon);
  swEnc     = false;
  if (partToken.Length()==0)
    _error = 6;
  else
  {
    swEnc = false;
    for (i=2;i<CFunVarLim&&!swEnc;i++)
    {
      if (NumToken==FunVarData[i].NumToken)
      {
        funvarLimits = FunVarData[i];
        swEnc=true;
      }
    }
    if (swEnc)
    {
      if (funvarLimits.IorF=="E")
        try {
          *_valor = partToken.ToInt();
        }
        catch (...) {
          _error =13; // Esperando número entero
        }
      else
        try {
          *_valor = partToken.ToDouble();
        }
        catch (...) {
          _error =14; // Esperando número entero
        }
      if (funvarLimits.lim_inf > (*_valor) || funvarLimits.lim_sup < (*_valor))
      {
        //
        // parametro fuera de rango
        //
        _error=15;
```

```
      } // (funvarLimits.lim_inf > valor2 || funvarLimits.lim_sup < valor2)
    }
    else  // (swEnc)
    {
      switch (NumToken)
      {
        case (120): // T
        {
          NHerr  = partToken.SubString(1,2).ToInt();
          NOS    = partToken.SubString(3,lon-2).ToInt();
          if (NHerr < 1 || NHerr > TotTools)
            _error = 46;
          else
            if (NOS < 1 || NOS > 20)
              _error = 47;
            else
            {
              *_valor  = NHerr;
              *_subind = NOS;
            }
          break;
        }  // case (120)
      }  // switch (Num)
    }  // else (swEnc)
  }  // if (partToken.Length()==0)
  return _error;
}
//---------------------------------------------------------------------------

//
// Recibe Num. de Token y la cadena del token que contiene el número
// si es U,W,X o Z devuelve el nuevo valor absoluto en BlkRec.U,BlkRec.W,
// BlkRec.X,BlkRec.Z para las demás variables solo devuelve el número de la
// cadena. Para I,K,R es necesario verificar el valor una vez que se ha
// identificado el G
//
int TCompilerForm::ChkAsgMovVar(typeNumToken NumToken,typeToken StrToken)
{
  int         _error,lon,numOffsetWork,numOffsetTool;
  float       cte,newValueF,newInc;
  AnsiString  partToken;

  _error    = 0;
  lon       = StrToken.Length()-1;
  partToken = StrToken.SubString(2,lon);
  if (partToken.Length()==0)
    _error = 6;
  else
  {
```

```
newValueF = partToken.ToDouble();
if (ActiveGValues[6]==21)  // sistema métrico
  cte = CCuentasMM;
else                       // sistema inglés
  cte = CCuentasPLG;
newValueF *= cte;
numOffsetWork = ActiveGValues[12] - 54;
numOffsetTool = PrgRec.TOS.value;
switch (NumToken)
{
  case (109): // I
  {
    BlkRec.I.value   = newValueF;
    BlkRec.I.actived = true;
    PrgRec.I.actived = true;
    break;
  }  // case (109) I
  case (111): // K
  {
    BlkRec.K.value   = newValueF;
    BlkRec.K.actived = true;
    PrgRec.K.actived = true;
    break;
  }  // case (111) K
  case (121):  // U
  {
    newInc   = newValueF;
    //
    // Para G04 no se verifica que U esté en el rango
    //
    if ( !(BlkRec.GWP.actived && BlkRec.GWP.value ==04) )
    {                              // excluir cuando sea G04
      newValueF = newValueF + PrgRec.X.value;
      if ((FunVarData[0].lim_inf * cte > newValueF) ||
         (FunVarData[0].lim_sup * cte < newValueF) )
      {
        _error=15;  // parámetro fuera de rango
      }
    } // if (!_error && !( .. && BlkRec.GWP.value ==04) )
    if (BlkRec.GWP.value ==04)
    {
      newInc = newValueF/cte;
    }
    if (!_error)
    {
      BlkRec.U.value   = newInc;
      BlkRec.U.actived = true;
    }
    break;
```

```
} // case (121) U
case (123): // W
{
  newInc   = newValueF;
  newValueF = newValueF + PrgRec.Z.value;
  if ((FunVarData[1].lim_inf * cte > newValueF) ||
     (FunVarData[1].lim_sup * cte < newValueF) )
  {
    _error=15; // parámetro fuera de rango
  }
  if (!_error)
  {
    BlkRec.W.value   = newInc;
    BlkRec.W.actived = true;
  }
  break;
} // case (123) W
case (124): // X
{
  newInc = newValueF;
  if (!_error && !(BlkRec.GWP.actived && BlkRec.GWP.value ==04) )
  {                              // excluir cuando sea G04
    newValueF = newValueF + OffsetWork[numOffsetWork].X
                  + OffsetTool[numOffsetTool-1].X;
    if ((FunVarData[0].lim_inf * cte > newValueF) ||
       (FunVarData[0].lim_sup * cte < newValueF) )
    {
      _error=15; // parámetro fuera de rango
    }
    else
    {
      newInc = newValueF - PrgRec.X.value;
    }
  } // if (!_error && !(... BlkRec.GWP.value ==04) )
  else
  {
    if (BlkRec.GWP.value ==04)
    {
      newInc = newValueF/cte;
    }
  }
  if (!_error)
  {
    BlkRec.X.value   = newInc;
    BlkRec.X.actived = true;
  }
  break;
} // case (124) X
case (126): // Z
```

```
      {
        newInc   = newValueF;
        newValueF = newValueF + OffsetWork[numOffsetWork].Z
                    + OffsetTool[numOffsetTool-1].Z;
        if ((FunVarData[1].lim_inf * cte > newValueF) ||
           (FunVarData[1].lim_sup * cte < newValueF) )
        {
          _error=15;  // parámetro fuera de rango
        }
        else
        {
          newInc = newValueF - PrgRec.Z.value;
        }
        if (!_error)
        {
          BlkRec.Z.value   = newInc;
          BlkRec.Z.actived = true;
        }
        break;
      }  // case (126) Z
      case (118):  // R
      {
        BlkRec.R.value   = newValueF;
        BlkRec.R.actived = true;
        PrgRec.R.actived = true;
        break;
      }  // case (118) R
    }  // switch (NumToken)
  } // if (partToken.Length()==0)
  return _error;
}
//---------------------------------------------------------------------------

int TCompilerForm::ChkActiveParameter(typeNumToken NumToken,typeToken StrToken)
{
  int   _error;
  float _valor;
  int   _subind;

  _error=0;
  switch (NumToken)
  {
    // estos errores hay que meterlos en una pila por que todavía no se
    // que G es el que estoy analizando, y el error de parámetro
    // inecesario para ese G tendría precedencia
    case (109) :   // I
    {
          //
          // I definido antes
```

```
      //
      if (BlkRec.I.actived)
        _error = 7;
      else
        _error = ChkAsgMovVar(NumToken,StrToken);
      break;
}
case (111) :
{
      //
      // K definido antes
      //
      if (BlkRec.K.actived)
      else
        _error = ChkAsgMovVar(NumToken,StrToken);
      break;
}
case (114) :
{
      //
      // N definido antes
      //
      if (BlkRec.N.actived)
        _error = 7;
      else
      {
        _error = ChkNumber(NumToken,StrToken,&_valor,&_subind);
        if (!_error)
        {
          if (topNLine != NULL)
          {
            if (_valor <= topNLine->NLine)
              _error = 43;
          }
          if (!_error)
          {
            if (!PushNLine(_valor))
              _error = 45;
          }
          if (!_error)
          {
            BlkRec.N.actived = true;
            BlkRec.N.value = _valor;
          }
        }
      }
      break;
}
case (116) :
```

```
{
      //
      // P definido antes
      //
      if (BlkRec.P.actived)
        _error = 7;
      else
      {
        _error = ChkNumber(NumToken,StrToken,&_valor,&_subind);
        if (!_error)
        {
          BlkRec.P.actived = true;
          BlkRec.P.value = _valor;
        }
      }
      break;
}
case (118) :
{
      //
      // R definido antes
      //
      if (BlkRec.R.actived)
        _error = 7;
      else
        _error = ChkAsgMovVar(NumToken,StrToken);
      break;
}
case (120) :   // T
{
   if (BlkRec.T.actived)
     _error = 7;
   else
   {

     _error = ChkNumber(NumToken,StrToken,&_valor,&_subind);
     if (!_error)
     {
       BlkRec.T.actived  = true;
       BlkRec.T.value    = _valor;
       BlkRec.TOS.actived = true;
       BlkRec.TOS.value  = _subind;

       PrgRec.T.actived  = true;
       PrgRec.T.value    = _valor;
       PrgRec.TOS.actived = true;
       PrgRec.TOS.value  = _subind;
     }
   }
```

```
    break;
}
case (121) :
{
      //
      // U definido antes
      //
      if (BlkRec.U.actived)
        _error = 7;
      else
        _error = ChkAsgMovVar(NumToken,StrToken);
      break;
}
case (123) :   // W
{
        _error = 7;// W definido antes
      else
        _error = ChkAsgMovVar(NumToken,StrToken);
      break;
}
case (124) :   // X
{
      if (BlkRec.X.actived)
        _error = 7;// X definido antes
      else
        _error = ChkAsgMovVar(NumToken,StrToken);
      break;
}
case (126) :   // Z
{
      if (BlkRec.Z.actived)
        _error = 7;// Z definido antes
      else
        _error = ChkAsgMovVar(NumToken,StrToken);
      break;
}
case (106) :   // F
{
      if (BlkRec.F.actived)
        _error = 7;// F definido antes
      else
      {
        _error = ChkNumber(NumToken,StrToken,&_valor,&_subind);
        if (!_error)
        {
          BlkRec.F.actived = true;
          BlkRec.F.value = _valor;
        }
      }
```

```
        break;
    }
    case (115) :   // O
    {
        if (BlkRec.O.actived)
          _error = 7;// O definido antes
        else
        {
          _error = ChkNumber(NumToken,StrToken,&_valor,&_subind);
          if (!_error)
          {
            BlkRec.O.actived = true;
            BlkRec.O.value = _valor;
          }
        }
        break;
    }
    case (119) :   // S
    {
        if (BlkRec.S.actived)
          _error = 7;// S definido antes
        else
        {
          _error = ChkNumber(NumToken,StrToken,&_valor,&_subind);
          if (!_error)
          {
            BlkRec.S.actived = true;
            BlkRec.S.value = _valor;
          }
        }
        break;
    }
  } // if (swEnc)
  return _error;
}
//---------------------------------------------------------------------------

//
// Esta función verifica que parámetros han sido activados y si al menos uno
// se proporciono determina que G es el que se intenta realizar
//

int TCompilerForm::ChkParameters(int *_nFunction, int *_cntMovParam, int *_cntOtherParam)
{
  int  swPos,i;
  int  _error;
  float speed;

  _error        = 0;
```

```
  (*_cntMovParam)  = 0;
  (*_cntOtherParam)= 0;
  *_nFunction      = -1;
//  numOffsetWork = ActiveGValues[12] - 54;
//  numOffsetTool = PrgRec.TOS.value;

  //
  // identifica que parámetros de movimiento están activos en el bloque
  //
  if (BlkRec.X.actived)      (*_cntMovParam)++;
  if (BlkRec.Z.actived)      (*_cntMovParam)++;
  if (BlkRec.U.actived)      (*_cntMovParam)++;
  if (BlkRec.W.actived)      (*_cntMovParam)++;
  if (BlkRec.R.actived)      (*_cntMovParam)++;
  if (BlkRec.I.actived)      (*_cntMovParam)++;
  if (BlkRec.K.actived)      (*_cntMovParam)++;
  //
  // identifica parámetros del bloque que no sean de Mov, ni: Gmov,GWP,T,S,F,O,N
  // que se activaron
  //
  if (BlkRec.P.actived)      (*_cntOtherParam)++;
  //
  // Identifica función activa
  //
  if (BlkRec.GWP.actived && BlkRec.GMov.actived)
    _error = 35;
  else
    if (BlkRec.GWP.actived)  // G c/parámetros pero no de movimiento activo
      *_nFunction = BlkRec.GWP.value;
    else
      if (BlkRec.GMov.actived)    // G de movimiento
        *_nFunction = BlkRec.GMov.value;
      else
        if ((*_cntMovParam) > 0)
        {
          if (PrgRec.GMov.value != 99)
          {
            *_nFunction = PrgRec.GMov.value;  // G de mov definido antes
            PrgRec.GMov.actived = true;
          else
            _error = 44;
        }
        else
          if ((*_cntOtherParam) > 0)
            _error = 32; // activando parámetros sin G
  //
  // Identifica parámetros de entrada de la función activa
  //
```

```
if (!_error)
{
  swPos = -1;
  for (i=0;i<CParamInOut && swPos==-1;i++)
  {
    if (Funciones[i].Function == *_nFunction)
      swPos = i;
  }
  if (swPos!=-1)
  {
    //
    // revisa que no se hayan proporcionado parámetros que no se
    // van a usar en la función activa
    //

    if (BlkRec.X.actived && !Funciones[swPos].DataIn.X)
      _error = 21;
    else if (BlkRec.Z.actived && !Funciones[swPos].DataIn.Z)
      _error = 22;
    else if (BlkRec.U.actived && !Funciones[swPos].DataIn.U)
      _error = 23;
    else if (BlkRec.W.actived && !Funciones[swPos].DataIn.W)
      _error = 24;
    else if (BlkRec.R.actived && !Funciones[swPos].DataIn.R)
      _error = 25;
    else if (BlkRec.I.actived && !Funciones[swPos].DataIn.I)
      _error = 26;
    else if (BlkRec.K.actived && !Funciones[swPos].DataIn.K)
      _error = 27;
    else if (BlkRec.X.actived && BlkRec.U.actived)
      _error = 28;
    else if (BlkRec.Z.actived && BlkRec.W.actived)
      _error = 29;
    else if (BlkRec.P.actived && !Funciones[swPos].DataIn.P)
      _error = 36;
    if (!_error)
    {
      if (BlkRec.GWP.actived)
      {
        if ( (*_cntMovParam) == 0 && BlkRec.GWP.value !=50 &&
            BlkRec.GWP.value !=97 )
        {
          _error = 30;
        }
        else
        {
          switch (BlkRec.GWP.value)
          {
            case (28):
```

```
        {
          PrgRec.G28.actived = true;
        }
        case (29):
        {
          if (!PrgRec.G28.actived)
            _error = 42;
          else
            PrgRec.G28.actived = false;
          break;
        }
        case (50):
        {
          if (!BlkRec.S.actived)
          {
            _error = 31;
          }
          else
          {
            BlkRec.G50.actived = true;
          }
          break;
        }
        case (97):
        {
          if (!BlkRec.S.actived)
          {
            _error = 31;
          }
          else
          {
            BlkRec.G97.actived = true;
          }
          break;
        }
      } // switch (BlkRec.GWP.value)
    } // else ( (*_cntMovParam) == 0 && BlkRec.GWP.value !=50 )
  }   // if (BlkRec.GWP.actived && !_error)
  else // if (BlkRec.GWP.actived)
  {
    if (*_nFunction>0 && *_nFunction<=3 &&
       ActiveMValues[2] != 3 && ActiveMValues[2] != 4 )
    {
      _error = 51;
    }
    else if (*_nFunction==2 ||*_nFunction==3)
    {
      // R tiene prioridad sobre I,K
      if (!PrgRec.R.actived && !(PrgRec.I.actived && PrgRec.K.actived))
```

```
        {
          _error = 49;
        }
        else // (!BlkRec.R.actived && !(PrgRec.I.actived ...
        {


        }   // (!BlkRec.R.actived && !(PrgRec.I.actived ...
      } // if (*_nFunction==2 ||*_nFunction==3)
    } // else (BlkRec.GWP.actived)
  } // if (!_error)
} // if (swPos!=-1)
//
// Valor de S
//
if (!_error && BlkRec.S.actived )
{
  if (BlkRec.G50.actived)
  {
     PrgRec.G50.actived = true;
     if (FunVarData[2].lim_sup < BlkRec.S.value)
       _error = 15;
     else
       PrgRec.G50.value   = BlkRec.S.value;
  } // if (!_error && BlkRec.S.actived )
  else
  {
    if (BlkRec.G97.actived)
    {
      PrgRec.G97.actived = true;
      PrgRec.G97.value   = BlkRec.S.value;
    } // if (!_error && BlkRec.S.actived )
    else
    {
      if (PrgRec.G50.actived)
      {
        speed = BlkRec.S.value / 300;
        if (FunVarData[2].lim_inf > speed)
        {
          _error=15;  // parámetro fuera de rango
        }
        else
        {
          if (speed > PrgRec.G50.value)
            speed = PrgRec.G50.value;
          if (!_error)
            BlkRec.S.value = speed;
        }
      }
```

```
            else
                _error = 50;
          } // else if (BlkRec.G97.actived)
        }
      } // if (!_error && BlkRec.S.actived )
    } // if (!_error)
    else

      *_nFunction = 0;

    return _error;
}

//
//  Actualizar parámetros que no generan bloque: F, O
//
int TCompilerForm::AssUniqueParameters(typeNumToken NumToken,typeToken StrToken)
{
    int _error,subind;
    float _valor;

    _error = 0;
    switch (NumToken)
    {
      case (106) :    // F
      {
        _error = ChkNumber(NumToken,StrToken,&_valor,&subind);
        if (!_error)
        {
          PrgRec.F.actived = true;
          PrgRec.F.value = _valor;
        }
        break;
      }
      case (115) :    // O, al inicio del programa, solo una vez y dentro del
      {               // los límites establecidos en el archivo de límites
        //
        // validacion para Ident. con O solo una ocasión
        //
        if (PrgRec.O.actived)
        {
          _error = 33;
        }
        else
        {
          //
          // validacion para Ident. con O al inicio del programa
          //
          if (TotBlks>0 || PrgRec.O.actived )
```

```
            // OR con todos aquellos parametros que no generan bloques
          _error = 34;
        else
        {
          _error = ChkNumber(NumToken,StrToken,&_valor,&subind);
          //
          // Ident. con O dentro de límites establecidos
          //
          if (!_error)
          {
            PrgRec.O.actived = true;
            PrgRec.O.value = _valor;
          }
        }
        break;
      }
    } // switch (NumToken)
  } // if (!_error)
  return _error;
}
//---------------------------------------------------------------------------

void TCompilerForm::IniVariablesBlk()
{
  int i;

  BlkRec.I.actived    = false;
  BlkRec.K.actived    = false;
  BlkRec.N.actived    = false;
  BlkRec.P.actived    = false;
  BlkRec.R.actived    = false;
  BlkRec.T.actived    = false;
  BlkRec.U.actived    = false;
  BlkRec.W.actived    = false;
  BlkRec.X.actived    = false;
  BlkRec.Z.actived    = false;

  BlkRec.F.actived    = false;
  BlkRec.S.actived    = false;

  BlkRec.GMov.value   = 99;
  BlkRec.GMov.actived = false;
  BlkRec.GWP.value    = 99;
  BlkRec.GWP.actived  = false;
  BlkRec.G50.value    = 0;
  BlkRec.G50.actived  = false;
  BlkRec.G97.value    = 0;
  BlkRec.G97.actived  = false;
  BlkRec.MFun.value   = 99;
```

```
  BlkRec.MFun.actived  = false;
  BlkRec.T.value       = 99;
  BlkRec.T.actived     = false;
  BlkRec.TOS.value     = 99;
  BlkRec.TOS.actived   = false;

  PrgRec.GMov.actived  = false;
  for (i=0;i<CPreFunc;i++)
  {
    MPreFunc[i].Actived=false;
  }
  for (i=0;i<CMisFunc;i++)
  {
    MMisFunc[i].Actived=false;
  }
}
//---------------------------------------------------------------------------

void TCompilerForm::IniVariablesPrg()
{
  PrgRec.X.value       = 0;    // Home Machine
  PrgRec.Z.value       = 0;    // Home Machine
  PrgRec.I.actived     = false;
  PrgRec.K.actived     = false;
  PrgRec.I.value       = 0;
  PrgRec.K.value       = 0;

  PrgRec.ExactSTP      = false;
  PrgRec.M30           = false;

  PrgRec.GMov.actived  = false;
  PrgRec.GMov.value    = 99;
  PrgRec.G28.actived   = false;
  PrgRec.G28.value     = 0;
  PrgRec.G50.actived   = false;
  PrgRec.G50.value     = 0;
  PrgRec.G97.actived   = false;
  PrgRec.G97.value     = 0;

  PrgRec.F.actived     = false;
  PrgRec.O.actived     = false;
  PrgRec.S.actived     = false;

  PrgRec.T.actived     = true;
  PrgRec.T.value       = 1;
  PrgRec.TOS.actived   = true;
  PrgRec.TOS.value     = 1;

  TotTools             = 20;
```

```
}
//---------------------------------------------------------------------------

//
// Para todas las llamadas a esta función el parámetro InpRec tiene
// la información de BlkRec con excepción de las funciones M, para ellas
// la información que se recibe es PrgRec
// InpRec tiene la información del bloque que se está procesando BlkRec
// o PrgRec para las funciones miscelaneas
//
// De última modificación:
//
// NÚMERO DE LINEA PARA IMPRIMIR EN PANTALLA   _line
//
int TCompilerForm::GenOutputRecord(int _NumG, DataRecord* InpRec, int _line)
{
  bool   swGenBlk[6];
  int    _error,numOffsetWork,numOffsetTool,i;
  float  valor,IncX,IncZ,radio,theta,incTheta,xm,zm;
  AnsiString Cadena[6];

  for (i=0;i<6;i++)
  {
    swGenBlk[i] = false;
    Cadena[i]   = "";
  }
  _error    = 0;
  numOffsetWork = ActiveGValues[12] - 54;
  numOffsetTool = PrgRec.TOS.value;

  switch (_NumG)
  {
    //
    // Generar bloque que posicionamiento o corte o sin parámetros
    //
    case 0:
    case 1:
    case 2:
    case 3:
    {
      //
      // VALIDACIONES PARA G de movimiento rápido o corte
      //
      // Spindle rotation normal or reverse, mordaza cerrada,
      // F definida, G50 activado
      //
      if (  _NumG != 0                                  &&
         !( (ActiveMValues[2] == 3  || ActiveMValues[2] == 4) &&
            ActiveMValues[4] == 11 && PrgRec.F.actived      &&
```

```
        PrgRec.G50.actived
      )
    )
  {
    if (!PrgRec.F.actived)
      _error = 39;
    if (!_error && ActiveMValues[4] != 11)
      _error = 38;
    if (!_error && !(ActiveMValues[2] == 3 || ActiveMValues[2] == 4))
      _error = 37;
    if (!_error && !PrgRec.G50.actived)
      _error = 41;
  }
  if (!_error)
  {
    swGenBlk[0] = true;
    swGenBlk[1] = true;
    swGenBlk[2] = true;
//
//      INCREMENTO en X
//
    if ((*InpRec).X.actived)
      IncX = (*InpRec).X.value; // incremento ya calculado con el offset
                                // con que fue generado en ChkAsg...
    else if ((*InpRec).U.actived)
        IncX = (*InpRec).U.value;
      else
        IncX = 0;
//
//      INCREMENTO en Z
//
    if ((*InpRec).Z.actived)
      IncZ = (*InpRec).Z.value; // incremento ya calculado con el offset
                                // con que fue generado en ChkAsg...
    else if ((*InpRec).W.actived)
        IncZ = (*InpRec).W.value;
      else
        IncZ = 0;
//
//      GENERA BLOQUES DE SALIDA PARA FUNCIONES DE CORTE Y MOVIMIENTO
//
    if (_NumG == 0)
    {
      Cadena[0] = "VS " + IntToStr(CFeedrateMax);
      Cadena[1] = "VP " + FloatToStrF(-IncX,ffFixed,8,0) + ","
                  + FloatToStrF(-IncZ,ffFixed,8,0);
      Cadena[2] = "BGS";
    }
    else if (_NumG == 1)
```

```
{
  Cadena[0] = "VS " + FloatToStrF(PrgRec.F.value*20,ffFixed,8,0);
  Cadena[1] = "VP " + FloatToStrF(-IncX,ffFixed,8,0) + ","
            + FloatToStrF(-IncZ,ffFixed,8,0);
  Cadena[2] = "BGS";
}
//
// Interpolación circular
//
else if (_NumG == 2 || _NumG == 3)
{
  if (IncX == 0 && IncZ == 0)
  {
    swGenBlk[0] = false;
    swGenBlk[1] = false;
    swGenBlk[2] = false;
  }
  else
  {
    theta     = 0;
    incTheta  = 0;
    radio     = 0;
    xm        = 0;
    zm        = 0;
    //
    // si xf,zf es el par ordenado 0,0 los puntos inicial y
    // final de la interpolacion pertenecen al mismo arco,
    // sino es así se realiza un arco del punto inicial al
    // par ordenado *_xf,*_zf y posterionemente una
    // interpolación lineal de *_xf,*_zf al punto final indicado
    //
    _error=GenCirInterp(_NumG,InpRec,IncX,IncZ,&theta,
                        &incTheta,&radio,&xm,&zm);
    if (!_error)
    {
      if (xm!=0 || zm != 0)
      {
        swGenBlk[3]=true;
        swGenBlk[4]=true;
        swGenBlk[5]=true;
        Cadena[0] = "VS " + FloatToStrF(PrgRec.F.value*20,ffFixed,8,0);
        radio = sqrt( (xm-PrgRec.X.value)*(xm-PrgRec.X.value) +
                    (zm-PrgRec.Z.value)*(zm-PrgRec.Z.value) );
        Cadena[1] = "CR " + FloatToStrF(radio,ffFixed,8,0) + ","
                    + FloatToStrF(theta,ffFixed,3,4) + ","
                    + FloatToStrF(incTheta,ffFixed,3,4);

        Cadena[2] = "BGS";
        Cadena[3] = "VS " + FloatToStrF(PrgRec.F.value*20,ffFixed,8,0);
```

```
            Cadena[4] = "VP "
               + FloatToStrF(-(PrgRec.X.value+IncX-xm),ffFixed,8,0) + ","
               + FloatToStrF(-(PrgRec.Z.value+IncZ-zm),ffFixed,8,0);
            Cadena[5] = "BGS";
          }    // if (xm!=0 || zm != 0)
          else  // if (xm!=0 || zm != 0)
          {
            radio = sqrt(IncX*IncX + IncZ*IncZ);
            Cadena[0] = "VS " + FloatToStrF(PrgRec.F.value*20,ffFixed,8,0);
            Cadena[1] = "CR " + FloatToStrF(radio,ffFixed,8,0) + ","
                      + FloatToStrF(theta,ffFixed,3,4) + ","
                      + FloatToStrF(incTheta,ffFixed,3,4);
            Cadena[2] = "BGS";

          } // else if (xm!=0 || zm != 0)
        }
        else //  if (!_error)
        {
          swGenBlk[0] = false;
          swGenBlk[1] = false;
          swGenBlk[2] = false;
        }
      }  // else if (IncX == 0 && IncZ == 0)
    }  // if (NumG == 2 || NumG == 3)
  } // primer if (!_error)
  if (!_error)
  {
    //
    // Actualiza los valores finales para las posiciones
    //
    PrgRec.X.value += IncX;
    PrgRec.Z.value += IncZ;
  } // segundo if (!_error)
  break;
} // case 3
case 4: {

  if ( (*InpRec).X.actived )
    valor = (*InpRec).X.value * 1000;
  else if ( (*InpRec).U.actived )
        valor = (*InpRec).U.value;
      else
        valor = 0;

  Cadena[0] ="WT " + FloatToStrF(valor,ffFixed,8,0);
  swGenBlk[0] = true;
  break;
}
case 28: {
```

```
//
// Genera bloque con valores a través de los cuales va a pasar
//
for (i=0;i<6;i++)
{
  swGenBlk[i] = true;
}
//
// Guarda los valores de la posición actual por si la llegara
// a utilizar con un G29
//
PrgRec.G28X.value = PrgRec.X.value;
PrgRec.G28Z.value = PrgRec.Z.value;
//
// Valor en el eje X
//
if ( (*InpRec).X.actived )
  IncX = - PrgRec.X.value + (*InpRec).X.value;
else if ( (*InpRec).U.actived )
      IncX = (*InpRec).U.value;
    else
      IncX = 0;
PrgRec.X.value += IncX;
//
// Valor en el eje Z
//
if ( (*InpRec).Z.actived )
  IncZ = - PrgRec.Z.value + (*InpRec).Z.value;
else if ( (*InpRec).W.actived )
      IncZ = (*InpRec).W.value;
    else
      IncZ = 0;
PrgRec.Z.value += IncZ;
Cadena[0] = "VS " + IntToStr(CFeedrateMax);
Cadena[1] = "VP " + FloatToStrF(-IncX,ffFixed,8,0) + ","
            + FloatToStrF(-IncZ,ffFixed,8,0);
Cadena[2] = "BGS";

//
// Evalua las dimensiones del movimiento necesario para
// regresar a la máquina a su home
// en PrgRec.G28Z.value se guardo la suma de los offset
// al momento del G28
//
IncX = -(PrgRec.X.value + OffsetWork[numOffsetWork].X
                + OffsetTool[numOffsetTool-1].X);
IncZ = -(PrgRec.Z.value + OffsetWork[numOffsetWork].Z
                + OffsetTool[numOffsetTool-1].Z);
PrgRec.X.value=0;
```

```
  PrgRec.Z.value=0;
  //
  // Genera bloque para regresar al home de la maquina
  //
  Cadena[3] = "VS " + IntToStr(CFeedrateMax);
  Cadena[4] = "VP " + FloatToStrF(-IncX,ffFixed,8,0) + ","
              + FloatToStrF(-IncZ,ffFixed,8,0);
  Cadena[5] = "BGS";
  break;
}
case 29: {
  //
  // Genera bloque con valores a través de los cuales va a pasar
  //
  for (i=0;i<6;i++)
  {
    swGenBlk[i] = true;
  }
  if ( (*InpRec).X.actived )
    IncX = (*InpRec).X.value-PrgRec.X.value;
  else if ( (*InpRec).U.actived )
        IncX = (*InpRec).U.value;
      else
        IncX = 0;
  PrgRec.X.value += IncX;
  //
  // Valor en el eje Z
  //
  if ( (*InpRec).Z.actived )
    IncZ = (*InpRec).Z.value-PrgRec.Z.value;
  else if ( (*InpRec).W.actived )
        IncZ = (*InpRec).W.value;
      else
        IncZ = 0;
  PrgRec.Z.value += IncZ;
  Cadena[0] = "VS " + IntToStr(CFeedrateMax);
  Cadena[1] = "VP " + FloatToStrF(-IncX,ffFixed,8,0) + ","
              + FloatToStrF(-IncZ,ffFixed,8,0);
  Cadena[2] = "BGS";
  //
  // Evalua las dimensiones del movimiento necesario para
  // regresar al punto anterior antes de enviar la máquina a
  // su home en PrgRec.G28X.value y PrgRec.G28X.value se guardo
  // el punto a donde hay que moverse
  //
  IncX = PrgRec.G28X.value - PrgRec.X.value;
  IncZ = PrgRec.G28Z.value - PrgRec.Z.value;
  PrgRec.X.value += IncX;
  PrgRec.Z.value += IncZ;
```

```
  //
  // Deja los valores de los ejes donde van a quedar
  //
  //
  //  Genera bloque para regresar al home de la maquina
  //
  Cadena[3] = "VS " + IntToStr(CFeedrateMax);
  Cadena[4] = "VP " + FloatToStrF(-IncX,ffFixed,8,0) + ","
              + FloatToStrF(-IncZ,ffFixed,8,0);
  Cadena[5] = "BGS";
  break;
}
case (119) :    // S
{

  swGenBlk[0] = true;
  Cadena[0]  = "OF ,," + IntToStr((*InpRec).S.value);
  break;
}  // end case 119 S
case (120) :    // T
{
  swGenBlk[0] = true;
  Cadena[0] = "Revolver " + IntToStr((*InpRec).T.value);
  break;
}  // end case 120 T
case (600) :    // M
{
  if ((*InpRec).MFun.value== 10 && ActiveMValues[2] != 5)
  {
    _error = 40;
  }
  else
  {
    switch ((*InpRec).MFun.value)
    {
      case (3) :
      {
        swGenBlk[0]=true;
        swGenBlk[1]=true;
        Cadena[0] = "CB4";
        Cadena[1] = "SB3";
        break;
      }
      case (4) :
      {
        swGenBlk[0]=true;
        swGenBlk[1]=true;
        Cadena[0] = "CB3";
        Cadena[1] = "SB4";
```

```
            break;
          }
          case (5) :
          {
            swGenBlk[0]=true;
            Cadena[0] = "CB1";
            break;
          }
          case (8) :
          {
            swGenBlk[0]=true;
            Cadena[0] = "SB5";
            break;
          }
          case (9) :
          {
            swGenBlk[0]=true;
            Cadena[0] = "CB5";
            break;
          }
          case (10) :
          {
            swGenBlk[0]=true;
            Cadena[0] = "SB2";
            break;
          }
          case (11) :
          {
            swGenBlk[0]=true;
            Cadena[0] = "CB2";
            break;
          }
          default :
          {
            swGenBlk[0] = true;
            Cadena[0]   = "M"+IntToStr((*InpRec).MFun.value);
            break;
          }
        } // switch ((*InpRec).MFun.value)
      } // if (!_error)
      break;
    }  // end case 600, M's
  } // switch
  //
  //  Graba bloque en campo memo
  //
  if (!_error)
  {
    MemoBlocks->Lines->BeginUpdate();
```

```
    for (i=0;i<6;i++)
    {
      if (swGenBlk[i])
      {
        TotBlks++;
        try
        {
          MemoBlocks->Lines->Add(Cadena[i]);
        }
        catch(...) {
          MemoBlocks->Lines->EndUpdate();  //se ejecuata en caso de excepción
          throw;
        }
        MemoBlocks->Lines->EndUpdate();  //se ejecuata en caso de excepción
      }
    } // for (i=0;i<6;i++)
  }
  return _error;
}

int TCompilerForm::GenCirInterp(int _numG, DataRecord *InpReg,float _incX,float _incZ,
      float *_theta, float *_IncTheta, float *_radio,float *_xm, float *_zm)
{
  int _error = 0;
  float  x1,x2,z1,z2;
  float  m,xc,zc,xf1,zf1,xf2,radioC,bRect,a,b,c,dist2;
  double theta, incTheta;
  //
  // pares coordenados 1,2 correpondientes a valores de inicio y fin de arco
  //
  x1 = PrgRec.X.value;
  x2 = PrgRec.X.value + _incX;
  z1 = PrgRec.Z.value;
  z2 = PrgRec.Z.value + _incZ;
  xf1 = 0;
  zf1 = 0;
  if (  BlkRec.R.actived ||
     (PrgRec.R.actived && !( BlkRec.I.actived || BlkRec.K.actived)) )
  {
    PrgRec.R.value = BlkRec.R.value;
    theta = atan(z1/x1);
    theta = theta * 180 / 3.14159265358979;
    incTheta = atan((z2-z1)/(x2-x1));
    incTheta = incTheta * 180 / 3.14159265358979;
    if (PrgRec.R.value < 0)
    {
      incTheta = 360 - incTheta;
    }
    if (_numG == 3)
```

```
  {
    incTheta = - incTheta;
  }
  *_radio = PrgRec.R.value;
}
else
{ // entonces deben estar activados los parámertros i y k
  if (BlkRec.I.actived)
    PrgRec.I.value = BlkRec.I.value;
  if (BlkRec.K.actived)
    PrgRec.K.value = BlkRec.K.value;
  if (!(PrgRec.I.actived && PrgRec.K.actived))
  {
    _error = 52;
  }
  else
  {
    radioC = PrgRec.I.value * PrgRec.I.value +
          PrgRec.K.value * PrgRec.K.value;
    xc = x1+PrgRec.I.value;
    zc = z1+PrgRec.K.value;
    dist2 = sqrt ( (z2-zc) * (z2-zc) + (x2-xc) * (x2-xc) );
    if ( (sqrt(radioC)) != dist2 )
    {
      m    = atan ((z2-zc)/(x2-xc));
      bRect = z2 - m*x2;
      a    = m*m + 1;
      b    = 2*m*bRect - 2*xc - 2*m*zc;
      c    = zc*zc + bRect*bRect - 2*zc*bRect - radioC;
      xf1  = ( -b + sqrt(b*b -4*a*c) ) / ( 2 * a);
      xf2  = ( -b - sqrt(b*b -4*a*c) ) / ( 2 * a);
      if (!(xf1>=xc && xf1<=x2) )
        xf1 = xf2;
      zf1 = m*xf1 + bRect;
      theta = atan(z1/x1);
      theta = theta * 180 / 3.14159265358979;
      incTheta = atan((zf1-z1)/(xf1-x1));
      incTheta = incTheta * 180 / 3.14159265358979;
      if (_numG == 3)
      {
        incTheta = - incTheta;
      }
    }
    else
    {
      theta = atan(z1/x1);
      theta = theta * 180 / 3.14159265358979;
      incTheta = atan((z2-z1)/(x2-x1));
      incTheta = incTheta * 180 / 3.14159265358979;
```

```
        if (_numG == 3)
        {
          incTheta = - incTheta;
        }
      }
    }
  }
  *_xm = xf1;
  *_zm = zf1;
  *_theta = theta;
  *_IncTheta = incTheta;

  return _error;
}

int TCompilerForm::PushNLine(int _NLine)
{
  ap_NLine *p;

  p = new ap_NLine [1];
  if (p == NULL)
  {
    MessageBox(Handle,"No hay suficiente memoria para buffer de NLinea", "Error Message",0);
    return(0);
  }
  else
  {
    p->NLine = _NLine;
    p->next  = topNLine;
    topNLine = p;
  }
  return(1);
}
//---------------------------------------------------------------------------

void TCompilerForm::FreeNLine()
{
  ap_NLine *p,*q;

  p = topNLine;
  //
  // q: apuntador a último nodo
  //
  while (p!=NULL)
  {
    q = p;
    p = p->next;
    delete[] q;
  }
```

```
  topNLine = NULL;
}
//---------------------------------------------------------------------------

int TCompilerForm::InsToken(typeNumToken _NumToken,typeToken _Token,int _NoLinea)
{
  ap_TokensTable *p,*q;

  p = new ap_TokensTable [1];
  if (p == NULL)
  {
    MessageBox(Handle,"No hay suficiente memoria para buffer de TOKEN", "Error Message",0);
    return(0);
  }
  if (headTokens == NULL)
  {
    headTokens = p;
  }
  else
  {
    q=headTokens;
    while (q->next!=NULL)
      q=q->next;
    q->next = p;
  }
  p->Token   = _Token;
  p->NumToken = _NumToken;
  p->NumLine  = _NoLinea+1;
  p->next    = NULL;

  return(1);
}
//---------------------------------------------------------------------------

int TCompilerForm::InsError(AnsiString _lErr,AnsiString _cErr,AnsiString _nErr, AnsiString
_dErr,AnsiString _idErr)
{
  ap_Errors *p,*q;

  p = new ap_Errors [1];
  if (p == NULL)
  {
    MessageBox(Handle,"No hay suficiente memoria para buffer de ERROR", "Error Message",0);
    return(0);
  }
  if (headErrors == NULL)
  {
    headErrors = p;
  }
```

```
    else
    {
      q=headErrors;
      while (q->next!=NULL)
           q=q->next;
      q->next = p;
    }
    p->lineaErr   = _lErr;
    p->columnaErr = _cErr;
    p->numErr     = _nErr;
    p->descErr    = _dErr;
    p->ident      = _idErr;
    p->next       = NULL;

    return(1);
}
//---------------------------------------------------------------------------

void TCompilerForm::FreeTokens()
{
  int row;
  ap_TokensTable *p,*q;

  StrGridTokens->RowCount=2;
  StrGridTokens->Cells[1][0]="Token";
  StrGridTokens->Cells[2][0]="NoToken";
  StrGridTokens->Cells[3][0]="Line";
  StrGridTokens->Cells[1][1]="";
  StrGridTokens->Cells[2][1]="";
  StrGridTokens->Cells[3][1]="";

  for (p=headTokens; p!=NULL;)
  {
    q=p->next;
    row = StrGridTokens->RowCount;
    StrGridTokens->Cells[0][row-1]=row-1;
    StrGridTokens->Cells[1][row-1]=p->Token;
    StrGridTokens->Cells[2][row-1]=IntToStr(p->NumToken);
    StrGridTokens->Cells[3][row-1]=IntToStr(p->NumLine);
    StrGridTokens->RowCount++;
    delete[] p;
    p=q;
  }
}
//---------------------------------------------------------------------------

void TCompilerForm::FreeErrors()
{
  int row;
```

```
  ap_Errors *p,*q;

  MainForm->StrGridErrors->RowCount=2;
  MainForm->StrGridErrors->ColWidths[4] = MainForm->Width -
              (MainForm->StrGridErrors->DefaultColWidth * 4);

  MainForm->StrGridErrors->Cells[0][0]= "Num.";
  MainForm->StrGridErrors->Cells[1][0]= "Lin.";
  MainForm->StrGridErrors->Cells[2][0]= "Col.";
  MainForm->StrGridErrors->Cells[3][0]= "Err.";
  MainForm->StrGridErrors->Cells[4][0]= "Descripción del Error";

  MainForm->StrGridErrors->Cells[1][1]="";
  MainForm->StrGridErrors->Cells[2][1]="";
  MainForm->StrGridErrors->Cells[3][1]="";
  MainForm->StrGridErrors->Cells[4][1]="";
  for (p=headErrors; p!=NULL;)
  {
    q=p->next;
    row = MainForm->StrGridErrors->RowCount;
    MainForm->StrGridErrors->Cells[0][row-1]=row-1;
    MainForm->StrGridErrors->Cells[1][row-1]=p->lineaErr;
    MainForm->StrGridErrors->Cells[2][row-1]=p->columnaErr;
    MainForm->StrGridErrors->Cells[3][row-1]=p->numErr;
    if (p->ident.Length()>0)
      MainForm->StrGridErrors->Cells[4][row-1]=p->descErr+" \""+p->ident+"\"";
    else
      MainForm->StrGridErrors->Cells[4][row-1]=p->descErr;
    MainForm->StrGridErrors->RowCount++;
    delete[] p;
    p=q;
  }
  if (MainForm->StrGridErrors->RowCount>2)
  {
    MainForm->StrGridErrors->RowCount--;
    MainForm->StrGridErrors->FixedRows=1;
  }
  headErrors = NULL;
  return;
}
//---------------------------------------------------------------------------

void TCompilerForm::ResultMsg(AnsiString _NameFile, AnsiString _CurrentLine,
                    AnsiString _TotalLines, AnsiString _FinalResult)
{
  CompileResultForm->LblCurrentLine->Caption = _CurrentLine;
  return;
}
//---------------------------------------------------------------------------
```

```
unsigned int TCompilerForm::Token(int* _line, int* _column, int* _lant, int* _cant,
                typeToken* _word, int _cnt_lines)
{
  bool sw_spc,sw_newline;
  char c;
  typeNumToken  NoEdo;
  AnsiString StrLine;

  NoEdo       = 0;
  (*_word)    = "";
  StrLine = TrimRight(CopyMemoEditor->Lines->Strings[*_line]);
  do { // mientras no encuentre linea con contenido
      sw_newline = false;
      if (StrLine==NULL)
        sw_newline = true;
      else
      {
        sw_spc=false;
        // salta espacios
        while ((*_column<=StrLine.Length()) && !(sw_spc))
          if (StrLine[*_column]==' ') (*_column)++; else sw_spc=true;
        if (*_column>StrLine.Length())
          sw_newline = true;
      }
      if (sw_newline && *_line < _cnt_lines)
      {
        (*_line)++;
        StrLine = TrimRight(CopyMemoEditor->Lines->Strings[*_line]);
      }
  } while (*_line < _cnt_lines && sw_newline);
  *_cant=*_column;
  *_lant=*_line;
  while ((*_word=="") && (NoEdo<100) && (*_line < _cnt_lines) )
  {
      // mientras no sea estado aceptor o fin de linea
      while (NoEdo<100 && (*_column<=StrLine.Length()))
      {
        c=UpCase(StrLine[*_column]);
        switch (UpCase(c))
        {
          case 'A':     NoEdo=MatTrans[NoEdo][0];  break;
          case 'B':     NoEdo=MatTrans[NoEdo][1];  break;
          case 'C':     NoEdo=MatTrans[NoEdo][2];  break;
          case 'D':     NoEdo=MatTrans[NoEdo][3];  break;
          case 'E':     NoEdo=MatTrans[NoEdo][4];  break;
          case 'F':     NoEdo=MatTrans[NoEdo][5];  break;
          case 'G':     NoEdo=MatTrans[NoEdo][6];  break;
          case 'H':     NoEdo=MatTrans[NoEdo][7];  break;
```

```
    case 'I':     NoEdo=MatTrans[NoEdo][8];  break;
    case 'J':     NoEdo=MatTrans[NoEdo][9];  break;
    case 'K':     NoEdo=MatTrans[NoEdo][10]; break;
    case 'L':     NoEdo=MatTrans[NoEdo][11]; break;
    case 'M':     NoEdo=MatTrans[NoEdo][12]; break;
    case 'N':     NoEdo=MatTrans[NoEdo][13]; break;
    case 'O':     NoEdo=MatTrans[NoEdo][14]; break;
    case 'P':     NoEdo=MatTrans[NoEdo][15]; break;
    case 'Q':     NoEdo=MatTrans[NoEdo][16]; break;
    case 'R':     NoEdo=MatTrans[NoEdo][17]; break;
    case 'S':     NoEdo=MatTrans[NoEdo][18]; break;
    case 'T':     NoEdo=MatTrans[NoEdo][19]; break;
    case 'U':     NoEdo=MatTrans[NoEdo][20]; break;
    case 'V':     NoEdo=MatTrans[NoEdo][21]; break;
    case 'W':     NoEdo=MatTrans[NoEdo][22]; break;
    case 'X':     NoEdo=MatTrans[NoEdo][23]; break;
    case 'Y':     NoEdo=MatTrans[NoEdo][24]; break;
    case 'Z':     NoEdo=MatTrans[NoEdo][25]; break;
    case ' ':     NoEdo=MatTrans[NoEdo][26]; break;
    case ';':     NoEdo=MatTrans[NoEdo][27]; break;
    case ':':     NoEdo=MatTrans[NoEdo][28]; break;
    case ',':     NoEdo=MatTrans[NoEdo][29]; break;
    case '.':     NoEdo=MatTrans[NoEdo][30]; break;
    case '+':     NoEdo=MatTrans[NoEdo][31]; break;
    case '-':     NoEdo=MatTrans[NoEdo][32]; break;
    case '*':     NoEdo=MatTrans[NoEdo][33]; break;
    case '/':     NoEdo=MatTrans[NoEdo][34]; break;
    case '=':     NoEdo=MatTrans[NoEdo][35]; break;
    case '%':     NoEdo=MatTrans[NoEdo][36]; break;
    case '#':     NoEdo=MatTrans[NoEdo][37]; break;
    case '(':     NoEdo=MatTrans[NoEdo][38]; break;
    case ')':     NoEdo=MatTrans[NoEdo][39]; break;
    case '[':     NoEdo=MatTrans[NoEdo][40]; break;
    case ']':     NoEdo=MatTrans[NoEdo][41]; break;
    default :
      if (c>='0' && c<='9')  NoEdo=MatTrans[NoEdo][42];
      else                   NoEdo=MatTrans[NoEdo][43];
  } // switch (UpCase(c))
  (*_column)++;
} // while (NoEdo<100 && (*_column<=StrLine.Length()))
if (NoEdo < 100) // fin de linea pero aun no es estado aceptor
{
  NoEdo=MatTrans[NoEdo][CColuMT-1];
  if ( NoEdo!=49 ) // ( detectando comentario
    *_word=StrLine.SubString(*_cant,(*_column)-(*_cant));
  if (NoEdo < 800)
  {
    *_column=1;
    (*_line)++;
```

```
          if (*_line < _cnt_lines)
            StrLine = TrimRight(CopyMemoEditor->Lines->Strings[*_line]);
        }
      } // (NoEdo < 100)
      else  // NoEdo >=100
      {
        // aquellos tokens que se identificaron por el siguiente caracter
        if ( NoEdo!=803) (*_column)--;
        if ( NoEdo<800 && NoEdo!=198 && NoEdo != 199) //  ; ó ( comentario )
          *_word=StrLine.SubString(*_cant,(*_column)-*_cant);
        if ( NoEdo==198 || *_column>StrLine.Length()  ||
           (NoEdo==199 && *_column==StrLine.Length())    )
        {
            (*_line)++;
            *_column=1;
            if ((*_line) <= _cnt_lines)
            {
              StrLine = TrimRight(CopyMemoEditor->Lines->Strings[*_line]);
            }
        } // if (NoEdo==198)
      } // else NoEdo<100
  } //     while ((*_word=="") && (NoEdo<300) && (*_line < _cnt_lines) )
  if (NoEdo == 49)
    NoEdo = 805;   // Error, comentario sin terminar
  return NoEdo;
}
//---------------------------------------------------------------------------

//
// Si hubo cambio de línea y no hay error, se ejecuta lo siguiente en el
// orden indicado, no importa el orden en que apareció en la línea:
//   - Se llama a ChkParameters, que revisa los parámetros activos y  genera
//     error si no corresponde los G con sus parámetros
//   - Si esta activo el parámetro S y no es la función G50, genera su bloque
//   - Si está activa la función de Herramienta genera su bloque
//   - De las funciones M activas en el bloque genera el orden las que se
//     activen al inicio del bloque
//   - Si hay función G activa, genera su bloque
//   - Genera las funciones M activas restantes las que se no van al inicio
//
void TCompilerForm::ChkChgLine(int _line, int _lant, int* _error)
{
  int cntMovParam,cntOtherParam,i;
  typeNumToken nFunc;

  //
  // si hubo cambio de línea y no ay error
  //
  if ((_lant != _line ) && !(*_error))
```

```
{
  *_error=ChkParameters(&nFunc,&cntMovParam,&cntOtherParam);
  if (!*_error)
  {
    if ( (BlkRec.S.actived) && (BlkRec.GWP.value!=50) &&
         (BlkRec.GWP.value!=97))
      *_error = GenOutputRecord(119,&BlkRec,_lant);
    if (!(*_error) && (BlkRec.T.actived))
      *_error = GenOutputRecord(120,&BlkRec,_lant);
    for (i=0;i<CMisFunc && !(*_error);i++)
    {
      if (MMisFunc[i].Actived && MMisFunc[i].AplicMoment==1 )
      {
        PrgRec.MFun.actived = true;
        PrgRec.MFun.value   = i;
        *_error = GenOutputRecord(600,&PrgRec,_lant);
        PrgRec.MFun.actived = false;
        PrgRec.MFun.value   = 99;
      }
    }
    for (i=4;i<CPreFunc && !(*_error);i++)
    {
      if (MPreFunc[i].Actived)
      {
        *_error = GenOutputRecord(MPreFunc[i].NumbFunc-CCtePreFunc,
                                  &PrgRec,_lant);
      }
    }
    if (!(*_error) && nFunc != -1 )
    {
      *_error=GenOutputRecord(nFunc,&BlkRec,_lant);
    }
    for (i=0;i<CMisFunc && !(*_error);i++)
    {
      if (MMisFunc[i].Actived && MMisFunc[i].AplicMoment!=1 )
      {
        PrgRec.MFun.actived = true;
        PrgRec.MFun.value   = MMisFunc[i].NumbFunc-CCteMisFunc;
        *_error = GenOutputRecord(600,&PrgRec,_lant);
        PrgRec.MFun.actived = false;
        PrgRec.MFun.value   = 99;
      }
    }
  }
}
if (*_error || _lant != _line )
{
  if (BlkRec.GMov.actived)
  {
```

```
      PrgRec.GMov.value   = BlkRec.GMov.value; // al principio tiene 99
    }                                 // cambia cada vez que se
    IniVariablesBlk();                      // actualice BlkRec.GMov.value
  }
}
//---------------------------------------------------------------------------

int TCompilerForm::PreFuncRecord(String token, PreparatoryFunctionRecord **R)
{
int i,RWFound;
  RWFound = false;
  for (i=0; i<CPreFunc && !RWFound ;i++)
    if ( MPreFunc[i].NameFunc == token )
    {
      *R =  &MPreFunc[i];
      RWFound = true;
    }
  return RWFound;
}
//---------------------------------------------------------------------------
int TCompilerForm::MisFuncRecord(String token, MiscellaneousFunctionRecord **R)
{
int i,RWFound;
  RWFound = false;
  for (i=0; i<CMisFunc && !RWFound ;i++)
    if ( MMisFunc[i].NameFunc == token )
    {
      *R =  &MMisFunc[i];
      RWFound = true;
    }
  return RWFound;
}
//---------------------------------------------------------------------------

int TCompilerForm::DescErrMess(int num, AnsiString *Msg)
{
int i,RWFound;
  *Msg="";
  RWFound = false;
  for (i=0; i<CErrMess && !RWFound ;i++)
    if (MErrMess[i].NumbErrM == IntToStr(num))
    {
      if (_language == 2)
        *Msg = MErrMess[i].DesSErrM;
      else
        *Msg = MErrMess[i].DesEErrM;
      RWFound = true;
    }
  return RWFound;
```

```
}
//---------------------------------------------------------------------------

void __fastcall TCompilerForm::FormClose(TObject *Sender,
    TCloseAction &Action)
{
  CompilerForm->Visible = false;
}
//---------------------------------------------------------------------------
```

```
//---------------------------------------------------------------------------
#ifndef CompileH
#define CompileH
#define CPreFunc        28      // Cantidad de funciones preparatorias 0 .. 27
#define CCtePreFunc     300     // Cte a sumar a Num. Func. Prep. para Num Pal Reserv
#define CMisFunc        30      // Cantidad de funciones miscelaneas  0 .. 29
#define CCteMisFunc     600     // Cte a sumar a Num. Func. Misc. para Num Pal Reserv
#define CErrMess        54      // Cantidad de errores             0 .. 53
#define CRowsMT         50      // Renglones de MT                 0 .. 49
#define CColuMT         44      // Columnas  de  MT                0 .. 43
#define CRowsMP         12      // Renglones de Parseo               0 .. 01
#define CColuMP          8      // Columnas  de Parseo               0 .. 07
#define CFunVarLim       7      // Limites para Funciones o Variables  0 .. 6
#define CGGroups        18      // Cantidad de grupos para func.prep.  0 .. 17
#define CMGroups         9      // Cantidad de grupos para func.misc.  0 .. 8, grupo 0 no se usa
#define CParamInOut     10      // Cant. de func. con param de In-Out  0 .. 9
#define COffsetTool     20      // Cantidad de offsets para herramienta
#define COffsetWork      6      // Cantidad de offsets de trabajo     G54..G59
#define CCuentasMM     1.5      // cuentas de encoder por milímetro
#define CCuentasPLG    1200     // cuentas de encoder por pulgada
#define CFeedrateMax 20000      // Feedrate máxima

//---------------------------------------------------------------------------
#include "CompileResultWin.h"
//---------------------------------------------------------------------------
#include<alloc.h>
#include <math.h>
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Db.hpp>
#include <DBGrids.hpp>
#include <DBTables.hpp>
#include <Grids.hpp>
//---------------------------------------------------------------------------
class TCompilerForm : public TForm
{
__published:    // IDE-managed Components
        TMemo *CopyMemoEditor;
        TStringGrid *StrGridTokens;
        TStringGrid *StrGridMisFunc;
        TStringGrid *StrGridPreFunc;
        TStringGrid *StrGridErrMess;
        TStringGrid *StrGridVarLimits;
        TMemo *MemoBlocks;
        void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
private: // User declarations

public:
    //
    // TIPOS PARA CARGAR TABLAS
    //
    // Campos para Funciones Preparatorias
    typedef struct {
      int Group;            // Grupo al que pertenece la funcion
```

```
  AnsiString NameFunc;     // nombre de la funcion
  int        NumbFunc;     // número de la función
  AnsiString DesEFunc;     // descripción en inglés
  AnsiString DesSFunc;     // descripción en español
  bool       Actived;
} PreparatoryFunctionRecord;
// Campos para Funciones Miscelaneas
typedef struct {
  int Group;               // Grupo al que pertenece la funcion
  int AplicMoment;         // Momento en el que se aplica 00 al inicio,99 final
  AnsiString NameFunc;     // nombre de la funcion
  int        NumbFunc;     // número de la función
  AnsiString DesEFunc;     // descripción en inglés
  AnsiString DesSFunc;     // descripción en español
  bool       Actived;
} MiscellaneousFunctionRecord;

typedef struct {
  AnsiString NumbErrM;     // número del error
  AnsiString DesEErrM;     // descripción en inglés
  AnsiString DesSErrM;     // descripción en español
} ErrWarMessageRecord;
//
// DEFINICION DE TIPOS PARA EL PROGRAMA
//
typedef AnsiString     typeToken;
typedef int            typeNumToken;

typedef struct {
  float value;
  bool  actived;
} ValFlt;

typedef struct {
  int   value;
  bool  actived;
} ValInt;

struct DIn{
  bool X;
  bool Z;
  bool U;
  bool W;
  bool I;
  bool K;
  bool R;
  bool P;
};

struct DOut{
  bool U;
  bool W;
  bool R;
};

typedef struct {
```

```
  bool  ExactSTP;
  ValFlt I;
  ValFlt K;
  ValFlt R;
  ValFlt U;
  ValFlt W;
  ValFlt X;
  ValFlt Z;
  ValFlt F;
  ValFlt G28X;
  ValFlt G28Z;
  ValInt P;
  ValInt N;
  ValInt O;
  ValInt S;
  ValInt T;
  ValInt TOS;
  ValInt GMov;
  ValInt GWP;  // G sin movimiento con parámetros
  ValInt G28;
  ValInt MFun;
  ValInt G97; //  valor de S que se envia multiplicado por la F activa
  ValInt G50;
  bool  M30;
  bool  ChgLine;
} DataRecord;

struct Param {
  struct DIn  DataIn;
  struct DOut DataOut;
  int       Function;
};

struct ap_TokensTable {
   typeToken    Token;
   typeNumToken  NumToken;
   int         NumLine;
   struct ap_TokensTable *next;
};

struct ap_Errors {
   AnsiString   lineaErr;
   AnsiString   columnaErr;
   AnsiString   numErr;
   AnsiString   ident;
   AnsiString   descErr;
   struct ap_Errors *next;
};

struct ap_NLine {
   int         NLine;
   struct ap_NLine *next;
};

//
// DEFINICION DE VARIABLE PARA EL PROGRAMA
```

```
//

ap_TokensTable *headTokens;
ap_Errors     *headErrors;
ap_NLine      *topNLine;

typedef struct {
   AnsiString Name;
   AnsiString PorN;  // P para aquellos parámetros que se utilizan para posicionamiento, N no
   AnsiString IorF;  // E para aquellos parámetros que deben ser enteros, D con decimales
   int  NumToken;
   Single lim_inf;   // limite inferior para la variable (mm)
   Single lim_sup;   // limite superior para la variable (mm)
} FunVarLimType; //    F,I,K,N,O,P,R,S,T,U,W,X,Z

FunVarLimType FunVarData[CFunVarLim];

struct {
   float X;
   float Y;
   float Z;
} OffsetTool[COffsetTool];

struct {
   float X;
   float Y;
   float Z;
} OffsetWork[COffsetWork];

int ActiveGValues[CGGroups];
int ActiveMValues[CMGroups];

DataRecord   BlkRec;
DataRecord   PrgRec; // sin usar I,K,P,R,U,W,X,Z,N,GWP

int   GBlkAct;
int   GMovAct;
int   GMovAnt;
int   TotBlks;
int   TotTools;
int   _language;
int   SValue,FValue,TValue;
int   ProgramValue,LineValue;

int   MatTrans[CRowsMT][CColuMT];
int   MatParseo[CRowsMP][CColuMP][4];

float  XPosition,ZPosition,UPosition,WPosition;
struct Param Funciones[CParamInOut];
AnsiString  CDir;
PreparatoryFunctionRecord   MPreFunc[CPreFunc];
MiscellaneousFunctionRecord MMisFunc[CMisFunc];
ErrWarMessageRecord         MErrMess[CErrMess];

//
// DEFINICION DE FUNCIONES
```

```
  //
    __fastcall TCompilerForm(TComponent* Owner); // User declarations
  int  OnCompile(AnsiString _NameFile);
  int  InsToken(typeNumToken _NumToken,typeToken _Token,int _NoLinea);
  int  PushNLine(int _NLinea);
  int  Parser(typeNumToken _NumToken,typeToken StrToken, int *_lParseo,
         int *_cParseo, PreparatoryFunctionRecord **RG,
                MiscellaneousFunctionRecord **RM);
  int  ActionsToGenCode(int _op,typeNumToken _NumToken,typeToken StrToken,
         int *_lParseo, int *_cParseo, PreparatoryFunctionRecord **RG,
                MiscellaneousFunctionRecord **RM);
  int  PreFuncRecord(String token, PreparatoryFunctionRecord **R);
  int  MisFuncRecord(String token, MiscellaneousFunctionRecord **R);
  int  DescErrMess(int num, AnsiString *Msg);
  int  DescWarMess(int num, AnsiString *Msg);
  int  ChkNumber(typeNumToken NumToken,typeToken StrToken, float *_valor, int *_subind);
  int  ChkAsgMovVar(typeNumToken NumToken,typeToken StrToken);
  int  ChkActiveParameter(typeNumToken NumToken,typeToken StrToken);
  int  ChkParameters(int *_nFunc,int *_cntMovParam, int *_cntOtherParam);
  int  GenOutputRecord(typeNumToken NumToken,DataRecord *InpReg, int _line);
  int  GenCirInterp(int _numG, DataRecord *InpReg,float _incX,float _incZ,float *_theta,
          float *_IncTheta,float *_radio,float *_xm, float *_zm);

  int  AssUniqueParameters(typeNumToken NumToken,typeToken StrToken);
  void IniVariablesBlk();
  void IniVariablesPrg();
  void FreeTokens();
  void FreeErrors();
  void FreeNLine();
  void ChkChgLine(int _line, int _lant, int *_error);
  void InsErrorWarning(int _error,AnsiString idErr,int *_cnt_errors,int _lant,int _cant);
  void ResultMsg(AnsiString _NameFile, AnsiString _CurrentLine,
         AnsiString _TotalLines, AnsiString _FinalResult);
  unsigned int Token(int* _line, int* _column, int* _lant, int* _cant, typeToken* _word, int _cnt_lines);
  int InsError(AnsiString _lErr,AnsiString _cErr,AnsiString _nErr, AnsiString _dErr, AnsiString _idErr);
};
//---------------------------------------------------------------------------
extern PACKAGE TCompilerForm *CompilerForm;

//---------------------------------------------------------------------------
#endif
```

SECRETARÍA DE EDUCACIÓN PÚBLICA

Sistema Nacional de Institutos Tecnológicos

Dirección General de Institutos Tecnológicos

CIIIEE'04

IV CONGRESO
Internacional sobre Investigación en Ingeniería Eléctrica y Electrónica

El Instituto Tecnológico de Aguascalientes
y el IEEE Sección Aguascalientes

Otorgan el presente

Reconocimiento

A: AURORA FEMAT DÍAZ

Por su Participación como:

PONENTE, CON EL TEMA **"Teoría de compiladores aplicada al diseño y desarrollo de un CNC"**

"IV Congreso Internacional sobre Investigación en Ingeniería Eléctrica y Electrónica"
celebrado del 15 al 19 de Noviembre del 2004

IEEE

Aguascalientes, Ags., México

Roberto Arana Morán
Director

José de Jesús López Villalobos
Presidente IEEE Aguascalientes

Teoría de compiladores aplicada al diseño y desarrollo de un CNC

Femat Díaz Aurora, Alaniz Lumbreras Daniel, Herrera Ruiz Gilberto,
Rico Hernández Ruth y Cervantes Pérez Sergio

Resumen--El presente trabajo muestra el desarrollo de la sección de software de un control numérico nacional, este consiste en el compilador y editor de dicho sistema, se describe la parte de ejecución del lenguaje de control numérico así como resultados del mismo. Se explican las etapas del proceso de compilación, así como la metodología empleada, que es flexible y puede ser aplicada a otros desarrollos. El lenguaje de control numérico usado por el compilador es código ISO G estándar y la salida generada será utilizada en una tarjeta de movimiento Galil.

Índice de Términos -- **CNC, Compilador, Máquina-Herramienta, Software.**

Nomenclatura -- **ISO Organización Internacional de Estándares, CNC Control Numérico Computarizado.**

INTRODUCCIÓN

El objetivo de este trabajo es el desarrollo del software necesario para la construcción de un sistema de CNC. El software incluye un editor, un compilador y un módulo de pruebas de directivas.

La mayoría de los CNC actuales, para ejecutar sus programas, utilizan intérpretes en lugar de compiladores. Un interprete verifica y ejecuta un programa línea a línea, interrumpiendo la ejecución del programa al encontrar el primer error (si lo hay), esto resulta inconveniente al estar maquinando una pieza; así mismo, dificulta el agregar procedimientos y paso de parámetros por lo que el lenguaje ISO de Gs se ha quedado muy retrasado ante el avance del software. Un compilador toma un programa como entrada y, si no encuentra errores en este, genera como salida otro programa en código apropiado para la máquina objeto, que puede correr de inicio a fin si su ejecución no se interrumpe por otro motivo; si se encuentran errores de compilación, se muestra una lista de todos ellos forzando a que el programa esté libre de errores antes de ser ejecutado. La presente metodología permite usar toda la capacidad del estándar ISO de Gs, realizar las validaciones que se consideren necesarias para los movimientos de los ejes en la máquina-herramienta o los mecanismos que la acompañan, así como ir creciendo en nuevas propuestas de funciones, apuntadores y otras capacidades que manejan los compiladores de lenguajes actuales. El sistema creado utilizará como antecedente las directivas básicas del trabajo de Herrera en [4].

I. BREVE DESCRIPCIÓN DEL SISTEMA

El sistema contiene un editor que permite tener varios programas en edición abiertos y uno solo activo; un compilador, cuya función es verificar las diferentes etapas de la compilación del programa activo e indicar los errores en dicho programa, o el éxito de la compilación; pantallas de captura para manipular los parámetros a ser utilizados en los programas, como offsets de trabajo y de herramienta, límites geométricos para los ejes, valores máximos para las velocidades, etc. y pantallas a través de las cuales sé monitorea el estado de los registros en el CNC una vez que este se encuentra ejecutando un programa.

La entrada del compilador es traducida en un programa objeto que contiene instrucciones que serán enviadas al controlador y a través de este a la máquina-herramienta; las directivas principales generadas para el controlador son interpolaciones lineales e interpolaciones circulares descritas en [3] a través de las instrucciones VP y CR respectivamente; las que especifican, a través de sus parámetros, los movimientos de los ejes en cuentas de encoder que el controlador enviará a la máquina-herramienta. Además, el programa de salida genera registros para aquellos códigos G ó M cuya función es activar o desactivar un estado en la máquina-herramienta, como por ejemplo: El abrir o cerrar la mordaza, la definición del sentido de rotación del husillo, encendido ó apagado de refrigerante, etc. y la configuración de parámetros de trabajo para la máquina-herramienta, como: la velocidad del husillo, la velocidad de alimentación por minuto ó por revolución, el número de herramienta a utilizar, etc.

II. ANTECEDENTES

En [5] se describe una máquina-herramienta CNC con tres unidades fundamentales:

1. La máquina-herramienta como tal, es decir, los elementos que permiten realizar el trabajo mecánico de arranque de viruta o desbaste, sujeción de las piezas de trabajo, los buriles y cortadores y la transmisión mecánica de potencia.
2. Unidades de potencia como motores eléctricos y amplificadores.

3. La unidad de CNC propiamente.

Al término control numérico se agrega el término computarizado, para indicar que se usa una computadora como medio para procesar la información numérica de entrada y convertirla en una salida apropiada para ser manejada por los controladores y actuadores del sistema, sustituyendo así mediante software tanto hardware como sea posible. La ventaja de este sistema es que al tener una etapa intermedia de compilación, entre el diseño del programa y su ejecución por el controlador, se validan las operaciones que se desea que la máquina-herramienta realice, de manera que la información puede ser verificada e incluso manipulada, tanto como sea necesario, antes de que esta sea interpretada por el controlador.

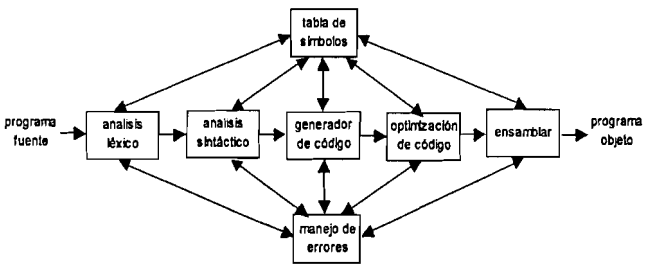
III. COMPILACIÓN

Un compilador es un programa que lee un código escrito en un lenguaje fuente y lo traduce a un programa en un lenguaje objeto [2]. La complejidad del lenguaje objeto puede variar desde un lenguaje de alto nivel hasta un lenguaje máquina dependiendo de la computadora destino, que puede ser desde un microprocesador hasta varias computadoras conectadas entre sí.

FASES DE UN COMPILADOR

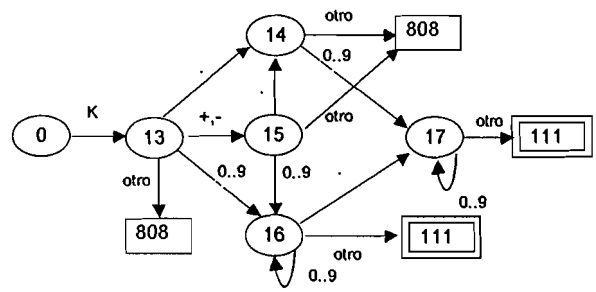
Conceptualmente, un compilador opera en fases, cada una de las cuales transforma al programa fuente de una representación a otra. En la práctica, se pueden agrupar algunas fases, como se hizo en este proyecto, y las representaciones intermedias entre estas fases no necesitan construirse explícitamente. En la Fig. 1.1 se muestran gráficamente las fases de un compilador.

Fig. 1.1 Fases de un Compilador



A. En el ***análisis léxico***, se leen las cadenas de caracteres que construyen el programa activo de izquierda a derecha y se agrupan en componentes léxicos que son secuencias de caracteres que tienen un significado colectivo, como por ejemplo: una letra G seguida de un número entero ó una letra I seguida de un número con punto decimal. Para especificar como se construyen dichos componentes del lenguaje, llamados tokens, se utiliza la herramienta de **diagramas de transición** basada en la teoría de autómatas [2], en la Fig. 1.2 se muestra un diagrama como ejemplo.

Fig. 1.2 Diagrama de Transición del Parámetro K



En dicho diagrama, se identifica una letra K seguida de un número con punto, con o sin signo, regresando un número 111 como número de token si el token fue identificado con un formato válido o un número 808 si ocurrió un error en dicha identificación.

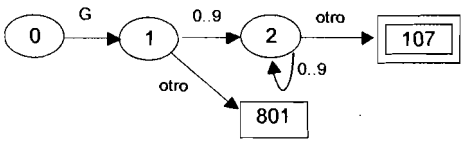
En el método utilizado para implementar los diagramas de transición todos los diagramas se llevan a una tabla, que también es llamada de transición. Este método consiste en crear una tabla en donde existe una fila por cada estado, una columna por cada símbolo de entrada y una última columna para los caracteres identificados como ``otro".

Para vaciar la información de los diagramas a la tabla, se toma en orden el primer diagrama que contienen el estado cero y la primer arista que sale de dicho estado; para cada uno de los caracteres de la arista se escribe en el número de columna asociado a dicho carácter: el número de estado de transición, el número de estado aceptor o el número de error al que apunta la arista; enseguida se toma la segunda arista que sale del estado inicial y se describe la información de los caracteres relacionados con ella, así, hasta terminar con las aristas en el primer diagrama. El proceso se repite para cada uno de los demás diagramas.

El proceso de exploración de la tabla se inicia en el renglón o estado cero. Al identificar el primer caracter se busca la columna definida para dicho caracter y se interpreta el contenido de la casilla como el renglón-estado hacia el cual se dirige el apuntador en la tabla de transición. Si el estado encontrado esta definido como aceptor, se termina de obtener el token. Si el estado encontrado está definido como de transición, este indica el nuevo renglón-estado para el apuntador en la tabla; se concatena el caracter encontrado; se realiza la lectura del siguiente caracter, quien define la columna donde se lee el siguiente estado-renglón; y se continua hasta encontrar un estado aceptor o un error. Si se encontró un estado aceptor este indica el número de token identificado y si se encontró un error, se le da un tratamiento a dicho error y se agrega a la lista de errores; en cualquier caso se continúa con el proceso de compilación.

En esta etapa se verifica también que cada una de las palabras identificadas formen parte del lenguaje; en el diagrama de transición de la Fig. 1.3, por ejemplo, se puede aceptar la letra G seguida de cualquier número entero para las funciones preparatorias, pero no todas esas palabras detectadas están implementadas en la máquina-herramienta utilizada, en casos como este también se agrega un error de léxico a la lista de errores.

Fig. 1.3 Diagrama de Transición de una Función Preparatoria



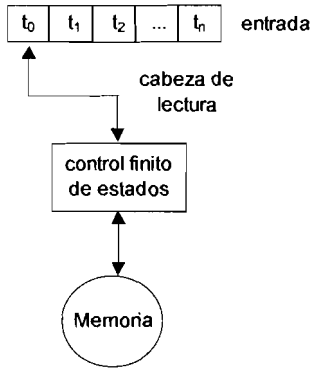
B. ***Análisis sintáctico***, en esta etapa los componentes léxicos se agrupan jerárquicamente en colecciones anidadas con un significado colectivo. Las secuencias de tokens se examinan para determinar si obedecen las convenciones de estructura explícitas en la gramática del lenguaje. En [1]-[2], se define a la gramática de un lenguaje como el conjunto de reglas que prescriben la estructura sintáctica de lenguajes bien formados.

En [1] se escriben dos métodos como los principales para definir lenguajes: **el generador y el reconocedor**. Aunque la mayoría de los lenguajes modernos de programación pueden ser descritos a través de generadores, en este trabajo se utilizó un reconocedor para definir el lenguaje.

En la implementación del análisis sintáctico se uso además una **tabla de parseo,** que es una estructura de datos que en los generadores contiene el conjunto de reglas de la gramática, esto, con la finalidad de establecer un proceso que facilite el agregar procedimientos e instrucciones con anidamientos en un futuro próximo.

La Fig. 1.4 muestra un diagrama de un reconocedor; el comportamiento de este se puede describir en términos de sus configuraciones. Una configuración es una fotografía del reconocedor describiendo los datos de entrada, el control finito de estados o el proceso que se realiza con los datos de entrada y la memoria o condiciones prerrequisito para obtener la salida del reconocedor. El reconocedor acepta una línea de entrada sí, a partir del estado del control, que en este proyecto se encuentra en la memoria del compilador, es posible llegar al estado deseado por la entrada; entonces, el reconocedor puede realizar una secuencia de pasos y terminar en una configuración final válida.

Fig. 1.4 Diagrama de un Reconocedor



El desarrollo de esta etapa se llevó a cabo de la siguiente manera: La fase de sintaxis solicita a la fase de léxico uno a uno los tokens de un bloque al mismo tiempo que navega a través de la tabla de parseo; cuando se recibe un token comienza el recorrido de la tabla de parseo en el primer renglón y primer columna; mientras se realizan las acciones de navegación, estas, cambian el renglón y la columna de ubicación del analizador de acuerdo al token recibido, hasta encontrar un error ó un estado aceptor. Después se solicita otro token con el que se vuelve a navegar a través de la tabla realizando las acciones que el recorrido indique, hasta encontrar un nuevo estado aceptor. Cuando se han verificado todos los tokens del bloque de entrada, el analizador sintáctico realiza aquellas revisiones que necesitan la información completa del bloque en memoria.

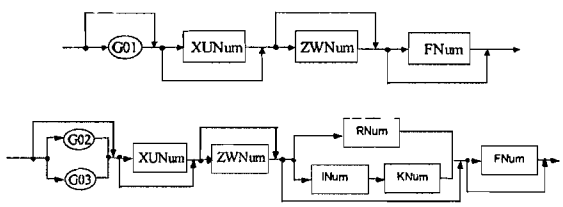
La tabla de parseo es un arreglo de tres dimensiones, con una profundidad de cuatro. La profundidad uno contiene la acción para navegar a través de la tabla y utiliza la información de las profundidades dos y tres para sus operaciones. Si la profundidad uno contiene una acción de comparación del número de token leído con el número indicado en la profundidad dos y esta comparación se cumple, entonces, se realiza la acción de sintaxis indicada en la profundidad cuatro y se salta a la columna indicada en la profundidad tres en el mismo renglón. Si no se cumple dicha condición, se continúa la navegación a través de la matriz en el mismo renglón y la siguiente columna. Las demás acciones de navegación a través de la tabla proporcionan un estado aceptor, saltan de renglón o generan un error de sintaxis.

Las acciones de sintaxis se dividen en dos grupos: las que se indican en la cuarta casilla de la matriz de parseo a través de un número y las que se llevan a cabo una vez que se tiene la información completa del bloque en proceso en la memoria. Las primeras se realizan al identificar el número de token esperado y revisan y actualizan el estado de la memoria durante el bloque. Las segundas relacionan la información del bloque analizado con el resultado de las funciones anteriores y son quienes verifican que la información sea suficiente para generar la(s) directiva(s) de salida que se pretenden realizar en el último bloque, si la información es insuficiente se genera un error.

Estos dos conjuntos de acciones trabajan como el control finito de estados del modelo del reconocedor. La base para diseñarlas fue: Analizar que información se requiere para generar cada directiva de salida, de que forma se define esa información y que funciones prerrequisito necesita. Luego se diseñó una estructura de datos que permite almacenar en memoria la información necesaria para validar cada una esas directivas de salida; se uso la tabla de parseo para poder escribir una acción después de identificar cada token y actualizar así la información relacionada con él preparando los datos para las revisiones al final del bloque; finalmente, se consideró la información para generar cada directiva de salida y sus funciones prerrequisito para diseñar las validaciones al final del bloque.

Como ejemplo, en la Fig. 1.5 se muestran dos diagramas para las instrucciones de interpolación lineal y circular respectivamente; estas instrucciones necesitan la ejecución previa de las funciones que se verifican a través de la operación número ocho de la Tabla A.2; cada uno de los parámetros en los diagramas debió ser almacenado en memoria durante el análisis del último bloque o en bloques anteriores de forma modal, indicando en el último bloque al menos un parámetro para especificar algún eje.

Fig. 1.5 Diagramas de las Funciones para Interpolación Lineal y Circular



Algunas de las acciones de sintaxis realizadas durante el análisis del bloque se describen en la Tabla A.1.

Tabla A.1 Algunas Acciones de Sintaxis

No.Acc.	Descripción
2	Verifica en la memoria qué el parámetro indicado por el token en análisis no haya sido definido antes en el mismo bloque; si ya había sido definido, genera un error de duplicidad de parámetro; si no, define el parámetro. Si se trata de un parámetro de corte o posicionamiento evalúa el valor de acuerdo al sistema de unidades y los offsets activos. Verifica que el parámetro esté dentro de los límites válidos. Si es un parámetro N, verifica que el valor dado sea menor al último proporcionado. Si no hubo algún error con las revisiones anteriores dentro de esta acción, asigna a la correspondiente variable del registro en memoria el nuevo valor.
3	Asigna el valor de entrada dado por el token a la variable del registro en la memoria y si se trata de la función O (nombre del programa), valida que sea la primer función generada.
5	Sí no ha sido activada en el bloque alguna función con parámetros, activa la función 4; si ya lo fue, genera el error 17.
6	Sí no ha sido activada en el bloque alguna función con parámetros, activa la función 28; si ya lo fue, genera el error 18.
10	Activa el indicador de salto de función (skip function).
11	Si está activa la variable del registro en memoria de paro exacto se desactiva, sino se activa.
12	Activa al grupo M al que pertenece la función con el número de M de la palabra y enciende el registro en memoria para generar el bloque de dicha función miscelánea.
13	Activa al grupo G al que pertenece la función con el número de G de la palabra.
14	Activa al grupo G al que pertenece la función con el número de G y enciende el registro en memoria para generar el bloque de dicha función preparatoria.
15	Activa la función para terminar el programa.

Las reglas del lenguaje tomadas como base en este proyecto, están basadas en el manual de programación CNC de Peter Smid [6]; el autor comenta que las palabras pueden estar en cualquier orden dentro de un bloque; por esto, fue necesario implementar revisiones hasta el final del bloque, algunas de estas operaciones se listan en la tabla A.2.

Tabla A.2 Algunas Operaciones al Final del Bloque

Rev.	Descripción
1	Si hay parámetros activos y no existe función en el bloque (o función modal especificada anteriormente) a la que puedan ser relacionados, se genera un error. Los parámetros de entrada se asocian a las funciones a través de una tabla que indica que variables pueden ser relacionadas con cada función preparatoria.
2	Si hay una función G28 se activa y se almacena el punto asociado a ella.
3	Si hay una función G29; verifica que se haya activado antes una función G28 (no necesariamente en el bloque anterior), si no fue así genera error.
4	Para cualquier función que necesite parámetros en el mismo bloque verifica que estos sean suficientes para poder generar el(los) bloque(s) de salida; si no lo son, genera error.
5	Se verifica que no se haya activado más de una función que requiere los mismos parámetros de entrada.
6	Si se especificó la función S y la función preparatoria G50, se establece el límite máximo de S para posteriores verificaciones.
7	Si se especificó S durante el bloque pero no para

	configurarla a través de G50, se verifica que el valor proporcionado en el bloque no exceda el límite anteriormente definido.
8	Si se va a realizar alguna función de corte se verifica que: la mordaza esté cerrada, se haya definido antes el sentido de rotación del husillo, que el valor de F (la velocidad de alimentación) haya sido especificado antes o en el bloque actual y que se haya generado un bloque definiendo el valor máximo para S a través de la función G50.

Si no se generó algún error con revisiones se termina la verificación de sintaxis del bloque de entrada. Algunos errores de sintaxis se muestran en la tabla A.3.

Tabla A.3 Algunos Errores de Sintaxis

Error	Descripción
7	Es necesario definir sentido de rotación para el husillo antes de algún corte.
26	Parámetro I no permitido para esta función.
28	Los parámetros X y U son excluyentes.
31	Esperando parámetro S.
33	Definiendo el nombre de programa más de una ocasión.
35	Definiendo más de un G con parámetros en un solo bloque.
36	Parámetro P no permitido para esta función.
38	Es necesario M11 para realizar algún corte.
40	Es necesario detener el husillo antes de abrir la mordaza.
42	No se ha activado G28 para la instrucción G29.
48	El valor de S excede el máximo especificado por G50.
49	No existe parámetro para especificar el radio.
50	Antes de definir S, es necesario que se especifique su valor máximo a través de G50.
52	El programa debe ser terminado a través de la función M30.

C. La fase de ***generación de código*** genera, a partir de las estructuras en memoria provenientes de un bloque y del estado en la memoria del compilador, secuencias de instrucciones de máquina que ejecutan las tareas indicadas en el bloque; se analiza el bloque de entrada y se genera una, más de una o ninguna directiva de salida resultado de las instrucciones analizadas y del estado de la memoria en el siguiente orden:

1) La función S (velocidad del husillo) si fue especificada sin G50 ó G97;

2) Cualquier cambio de herramienta;

3) Las funciones misceláneas activadas que deben ser enviadas al inicio del bloque;

4) Las funciones preparatorias que generan bloque de salida pero que no son de movimiento o de corte;

5) La función de posicionamiento o corte y

6) Las funciones misceláneas que se aplican durante o al final del bloque.

Para las funciones preparatorias que tienen salida con parámetros se describe el contenido de su directiva a continuación:

1) Para la función G00 se especifican los incrementos con signo en cuentas de encoder de la última posición a la posición definida en el bloque de entrada para cada uno de los ejes y se indica si el posicionamiento es con paro exacto o sin el.

2) Para la función G01 se especifican los incrementos con signo en cuentas de encoder de la última posición a la especificada en el bloque de entrada para cada eje, si es con paro exacto o no y la velocidad de alimentación definida para el corte.

3) Para G02 y G03 se proporciona el punto final de la interpolación con incrementos en cuentas de encoder para cada uno de los ejes, el ángulo que forma el punto actual con respecto al origen de la máquina, el incremento del ángulo en radianes con signo para llegar al punto final de la interpolación y la velocidad de alimentación definida para el corte.

Si es una interpolación circular en sentido horario (G02) o antihorario (G03), queda explícita en el sentido de los ángulos especificados. La interpolación circular puede estar compuesta también de dos bloques de salida, esto depende de sí los parámetros del arco de entrada son exactos, si es así, será solo un bloque; si no lo son y ocurre que el centro del arco fue especificado a través de los parámetros I,K y el punto final del arco no es equidistante del centro de este, se generan dos bloques: El primero de ellos como una interpolación circular del punto actual al punto formado por la intersección de la circunferencia con el centro dado por I,K y la recta que va del centro de circunferencia al punto final especificado y el segundo bloque para la interpolación lineal del punto de intersección al punto final.

4) Para G04 se describe el valor de la pausa en milisegundos.

5) Para G28 se generan dos bloques de salida con parámetros de incremento en cuentas de encoder en cada eje; el primer bloque es para realizar el desplazamiento del punto actual al punto intermedio especificado por la instrucción de entrada y el segundo bloque para el movimiento del punto intermedio al origen de la máquina.

6) Para G29 también se generan dos bloques de salida con parámetros de incremento en cuentas de encoder; el primer bloque es para realizar el desplazamiento del punto actual al especificado por la instrucción de entrada y el segundo bloque para el movimiento del punto intermedio al especificado a través de la función G28 relacionada con el G29 en proceso.

7) Para las funciones G50 y G92 se especifica el valor de S como la cantidad máxima de revoluciones por minuto.

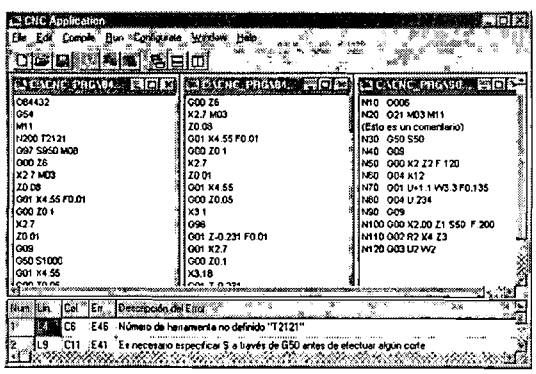
8) Para G97 se configura la entrada en revoluciones por minuto, cancelando así G96.

También existen algunas funciones G que le indican al controlador la activación o desactivación de alguna función de la máquina herramienta, aunque no necesitan parámetros, ellas requieren que se genere su bloque de salida, las demás funciones preparatorias son usadas solo por el compilador.

Una función M (miscelánea) en la salida representa la activación o desactivación de un dispositivo en la máquina-herramienta.

Sabemos que todas las fases del proceso de compilación se realizan de manera conjunta, de manera que, con una pasada se efectúa la compilación completa del programa fuente, mientras se están realizando las etapas de análisis también se está llevando a cabo la generación de código. En la Fig. 1.6 se muestra un ejemplo de una pantalla de compilación.

Fig. 1.6 Una pantalla de compilación
de un programa con errores



D. **Detección e información de errores**. Cada fase puede encontrar errores y al detectar uno de ellos se le da un tratamiento a ese error para poder continuar con la compilación, permitiendo así la detección de más errores en el programa fuente [2].

Por lo general las fases de análisis léxico y sintáctico, manejan la mayoría de los errores detectados por un compilador. La fase de léxico detecta los errores donde un conjunto de los caracteres en un bloque no forman algún componente válido del lenguaje. Los errores donde la secuencia de componentes léxicos o palabras en un bloque no cumplen las reglas de estructura del lenguaje fueron determinados por la fase de análisis sintáctico.

Algunas características adicionales del sistema:

1) Se manejan 20 offsets de trabajo, modificables a través del sistema de manera automática, como lo hacen los CNC del mercado.
2) Se manejan offsets de herramienta también de manera automática y modificables.
3) Se validan los límites geométricos de los ejes.
4) Se validan las velocidades de alimentación o del husillo para la máquina-herramienta de manera automática.
5) El sistema permite agregar instrucciones de manera sencilla.

Plan a corto plazo:

6) Manejar compensaciones, manipuladas directamente por el compilador antes de llegar a la máquina-herramienta;
7) Ciclos;
8) Instrucciones condicionales y
9) Llamada a procedimientos.

A largo plazo:

10) Manejo de diferentes idiomas y
11) Diferentes máquinas-herramientas en un mismo programa.

Reconocimientos

A CONACYT por la beca proporcionada al primer autor para la realización de sus estudios de maestría.

Referencias

[1] Aho, Alfred y Ullman, Jeffrey. Compilers-Principles, Techniques and Tools. Addison Wesley, 1986.

[2] Aho A., Sethi R. y Ullman J. ``The Theory of Parsing, Traslation and Compiling". Prentice Hall, 1972.

[3] Galil Motion Control, Inc. Command Reference Manual Rev 1.0a, Rev. Abril 1999.

[4] Herrera, Gilberto. Sistema de Control para Máquinas-Herramienta CHROM-II, Certificado de derechos de autor, Instituto Nacional de Derechos de Autor, 2004.

[5] Herrera, Gilberto. A Modular CNC Controller, Dissertation Submitted for the Doctoral Degree in Engineering, Computer and Automation Institute, Hungarian Academy of Sciences, Department of Manufacturing Engineering, Technical University of Budapest, Budapest, Hungary, 1992.

[6] Smid, Peter. CNC Programming Handbook, 2nd Edition, Industrial Press, Inc., NY, USA, 2003.

Aurora Femat Díaz
exterra@prodigy.net.mx
Ingeniería en Sistemas Computacionales 89-94, UAA
Actualmente:
Estudiante de Maestría en Instrumentación y Control Automático
Universidad Autónoma de Querétaro
Facultad de Ingeniería
Cerro de las Campanas s/n,
Querétaro Qro. México