A Compiler for NC Parametric G-Code Programs

Project Report

Submitted by

R. VISHNU DEV
E. M. NALINADEVI
P. MADHUKAR

Guided by

Smt. S. DEVAKI, B.E., M.S., M.I.S.T.E.

1995 - 96

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF **BACHELOR OF ENGINEERING IN COMPUTER SCIENCE AND ENGINEERING** OF BHARATHIAR UNIVERSITY, COIMBATORE

Department of Computer Science and Engineering

Kumaraguru College of Technology

Coimbatore - 641 006

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Kumaraguru College of Technology

COIMBATORE - 641 006

CERTIFICATE

This is to Certify that the report titled

A COMPILER FOR NC PARAMETRIC G - CODE PROGRAMS

has been submitted by

R. VIJ[illegible]

[illegible]

Mr. / Ms. [illegible]

In partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING IN THE COMPUTER SCIENCE AND ENGINEERING BRANCH

of the Bharathiar University, Coimbatore-641 046

during the academic year 1995 - 96.

Guide

Head of the Department

Certified that the candidate was examined by us in the project work Viva-voce held on [illegible]

and the University Register Number is [illegible]

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

This project sailed through from concept to completion due to the immense help rendered by many.

Our mentor and guide, **Smt. S.Devaki**, B.E.,M.S., M.I.S.T.E., was our pillar of support. She provided, with consummate ease,accurate and timely solutions to the multifarious problems.

We are forever indebted to **Prof.P.Shanmugam**, M.Sc(Engg), M.S(Hawaii), Sr.M.I.E.E.E., M.I.S.T.E., Head of the Department of Computer science and Engineering, for allowing us to embark on the project.

We express our sincere thanks to **Dr.S.Subramanian**, B.E., M.Sc.(Engg), Ph.D., Principal, Kumaraguru College of Technology, for instilling the Principles of Software Engineering in us.

Dr.Babu Padmanabhan, M.D.,Steer Engg (P) Ltd.,for whose factory we did the Project, mooted the ideas, motivated us and mobilised the necessary support for us. Able support, succor and encouragement was forthcoming in abundance from **Sri. Satish Padmanabhan**, A.C.A., Grad. C.W.A., Financial Controller, Steer Engg(P)Ltd. Our thanks to them cannot be expressed in mere words.

We thank members of the Faculty and our friends, who provided us much needed support at every stage.

SYNOPSIS

The project, " **A compiler for NC parametric G-code programs**",was developed at Kumaraguru College of Technology for Steer Engineering (P) Limited. The compiler was developed using Turbo C++, with a user interface under Visual Basic.

The compiler accepts as input, a parametric G-code program. After performing the compilation process, it generates as output,a standard G-code file which can be simulated by commercially available simulation packages such as NCPACK.

CONTENTS

Note : Details of the MC 8600 numerical control & program codes are given in a separate appendix.

STEER ENGINEERING PRIVATE LIMITED : A PROFILE

Steer Engineering Private Limited is engaged in the manufacture of import substitutes for the Engineering Plastics Industry. The components manufactured are technically called "Twin-Screw Compounding elements".

These are manufactured at their sophisticated CNC Shop which includes the company's in-house CAD/CAM facility. The factory is located in an Industrial Estate in Peenya, Bangalore.

The major items manufactured by Steer Engineering (P) Ltd are :

a. Kneading Blocks

b. Conveying Screws

c. Side-feeding Screws

d. Barrel Liners

e. Barrels

The CNC facility consists of :

1. CNC Vertical Machining Center

2. CNC Wire-cut Electric Discharge Machine

A photograph of the machining facility for which we designed the compiler is attached in the following sheet.

INTRODUCTION

G-code : G-code is the programming language used to run CNC machines. G-code programs are usually directly written by CNC programmers (or) generated with the help of appropriate software. Due to tremendous human involvement necessary, there is scope for error.

Debugging G-code before machining requires a tool that checks the program for errors and simulates the operation of the machine for a particular G-code program. In fact, most CNC machines have graphics capability that allows simulation of the program before actual cutting. However, the machine is kept idle during this time and if errors are found, corrections cause further wastage of machine time. This leads to improper utility of the machine.

In order to maximize utility of the CNC machine, a compiler that can convert the parametric G-code into a form that can be simulated by commercially available simulating software, needs to be available to the CNC programmer during code generation.

A standard G-code is fairly simple. Apart from the standard G-code,different control systems offer tools such as parametric programming to enhance the programming capability. Parametric programming allows the use of variables, assignment statements and loop operations, very similar to high level languages.

Writing a compiler for a parametric programming language is similar to writing a compiler for a high level language like "C".

NEED AND OBJECTIVE

The objective of this project is to design,develop and implement a compiler which converts the parametric G-code program of the MC8600 numerical control into a standard G-code program. This standard form can be simulated by commercial simulators, whereas the parametric program cannot be simulated by them.

The **features** of such a tool are:

1. Ability to deal with arithmetic expressions.
2. Ability to deal with loop statements.
3. Error detection capability.

The **advantages** of such a tool are:

1. It enables off-line simulation of the G-code program possible, thus increasing the productivity of the machine.
2. It enables program analysis, to maximize machining time and reduce idle cutter passes.
3. It enables error detection and debugging in the code generation stage itself.

This project was envisaged with an aim to fulfill the above requirements, as specified by DR.Babu Padmanabhan, Managing Director, Steer Engg(P) Ltd.

REQUIREMENTS ANALYSIS

After detailed discussion with the people at Steer Engineering, we obtained a clear idea that what they desired to have was a tool to perform the following tasks:

1. **Program analysis**

The program submitted to the machine often has errors of various kinds. A good compiler therefore , should detect the errors. With regard to the errors, the compiler must be an effective communication device;it should deliver comments to the user in a language and in the manner the user can understand.

The typical lexical errors possible are :

* Insertion of an extraneous character.
* Deletion of a required character.
* The replacement of a correct character by an incorrect character or token.
* The transposition of two adjacent characters or tokens.

It is convenient to classify errors as either syntactic or semantic.We define syntactic error to be an error detectable by lexical or syntactic face of a compiler. Other errors detectable by the compiler are termed as semantic errors. The common semantic errors are undeclared or multiply-declared identifiers. Type incompatibilities between operators and operands and between formal and actual parameters are another common source of semantic errors that can be detected in many languages at compilation time.

2. Conversion of parametric G-code to ISO G-code:

The primary concern was the utility of the machine, as machine time is directly related to production and delivery schedules. This meant that the programs needed to be simulated before being run on the machine. Though, commercial software for simulating ISO G-code was available, these were incapable of accepting statements unique to the MC8600 control system. Hence these statements needed to be changed into standard G-code.

The above requirement lead to a host of smaller requirements to be met, such as:

1. Variables used had to be replaced by their actual values.
2. Loop structures need to be removed.
3. Subroutines had to be integrated into the main program.
4. A host of statements unique to MC8600 control system needed to be analysed as to what they intended to do and converted accordingly into the ISO G-code format.

SYSTEM OVERVIEW

The process of compilation is a complex one. Therefore, it is necessary to partition the compilation process into a series of sub processes called phases.

A phase is a logically cohesive operation that takes as input one representation of the source program and produces as output another representation. Our compiler has been constructed along the lines specified by conventional compiler design procedure. The phases involved are shown in Figure 1.

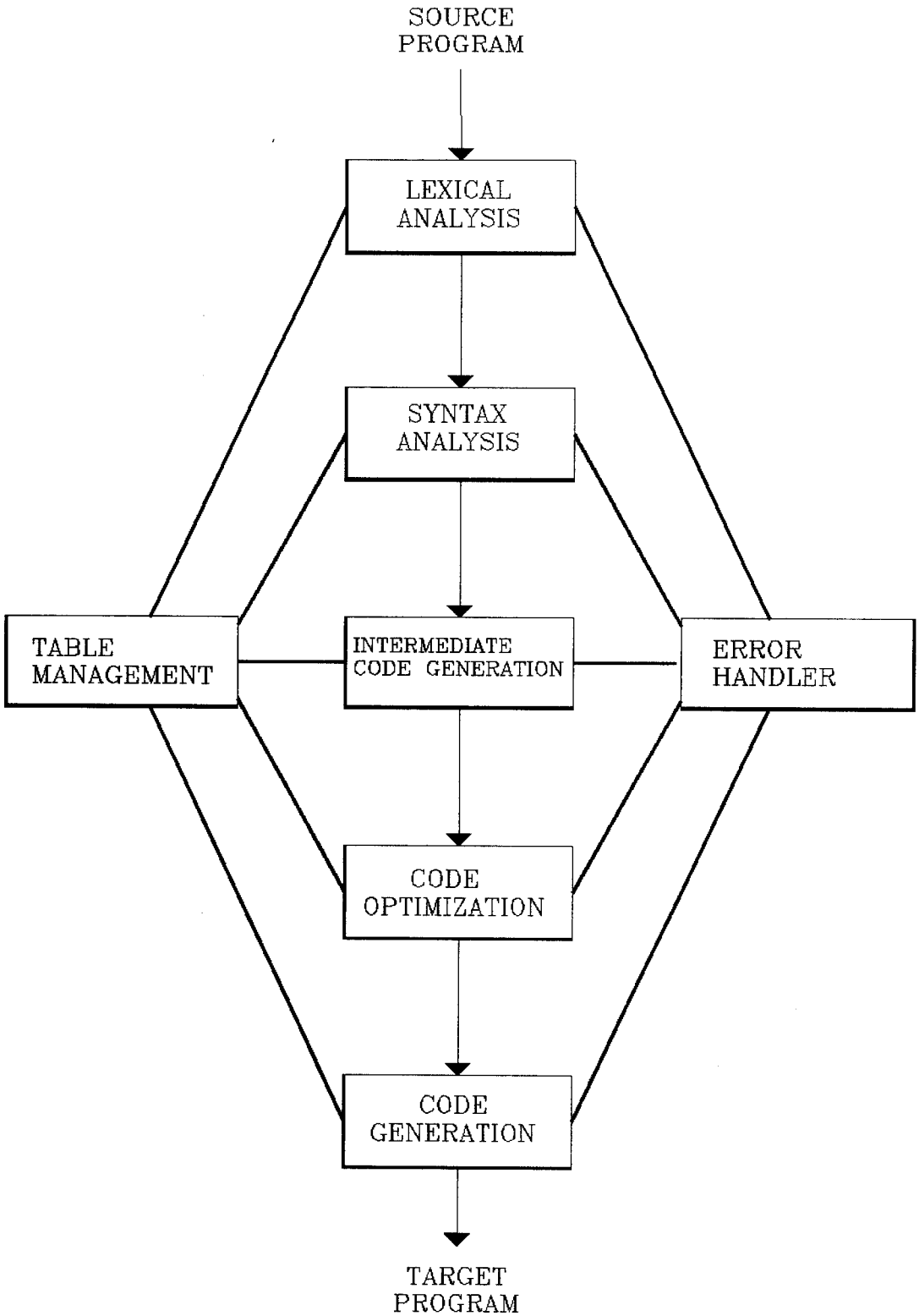


FIGURE 1. PHASES OF A COMPILER

The lexical analyser scans the input program line by line and identifies and stores valid tokens. After the lexical analyser has determined that the line does not contain invalid tokens, the syntax analyser verifies that the line is syntactically correct.

The intermediate code generation phase occurs in parallel with lexical and syntax analysis. The intermediate code is a collection of parsed token strings.

The code generation phase consists of evaluating the expressions. The target program specifies only the G statements along with the x,y,z co-ordinate values,along with the rotational axis,"a". The table management portion of the compiler maintains a symbol table, which is a list of variables and their current values. The symbol table is created during lexical analysis and the values of the variables are computed during expression evaluation.

The error handler is invoked whenever a flaw is detected during compilation and the corresponding message is given to the user.

In the implementation of the compiler, the lexical and syntax analysis phases are combined into a pass,which produces the intermediate code. In the second pass, the code generator scans the intermediate code, evaluates the expression and outputs the target program. The target program is standard G-code,which can be simulated.

SYSTEM DESIGN

ARCHITECTURAL DESIGN

By architectural design, we have developed a modular program structure and represented the control relationships between modules. As specified in the requirement analysis, the lexical analyser and the parser perform the task of program analysis .

The code generator performs the task of converting the parametric G-code to standard G-code.

The data flow diagram is shown in Figure 2.

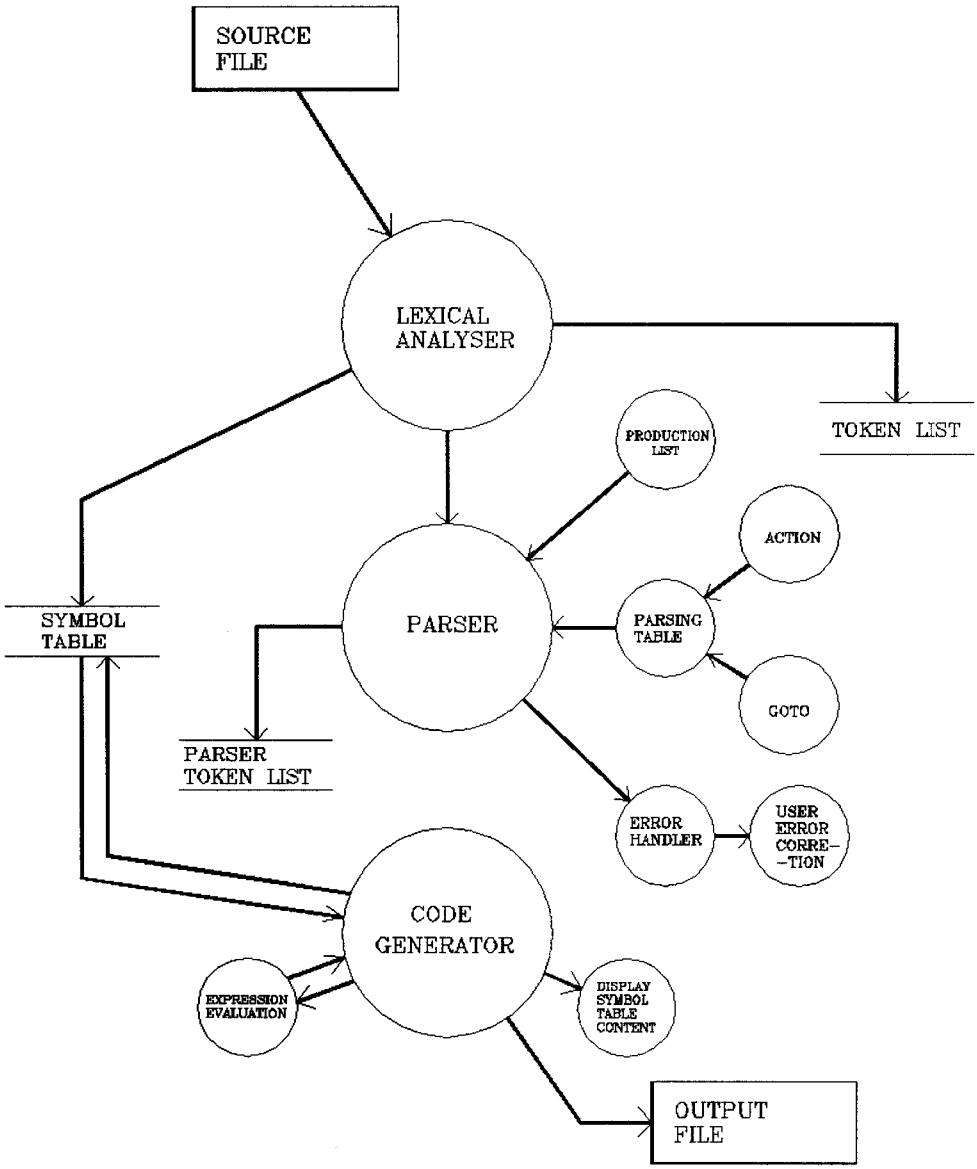


FIGURE 2. DATA FLOW DIAGRAM

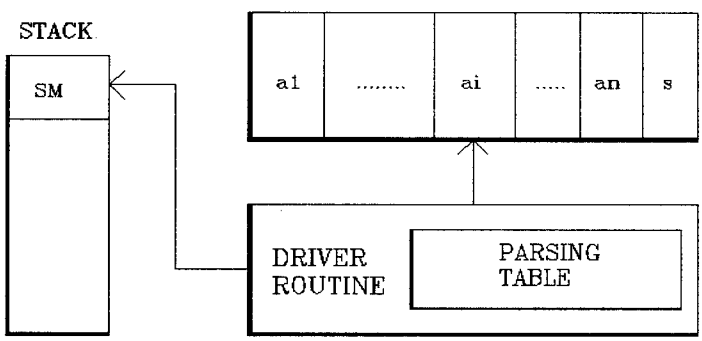


FIGURE 3. L R PARSER

DETAILED DESIGN

The detailed system design comprised of designing each component of the compiler.

1. DESIGN OF THE LEXICAL ANALYSER

The main purpose of a lexical analyser is to translate the input string into a form that is more manageable by the parser. It translates input strings or lexemes into tokens-arbitrary integer values that represent the lexemes. A token has a one - one relationship with the lexeme.

Lexical analysers often have auxiliary functions as well,for example, they have to skip over white spaces, they have to keep track of the correct line number, etc.

Another function of the lexical analyser is to translate numeric constants stored in ASCII format into the associated number.

1. a) SYMBOL TABLE MANAGEMENT

When an identifier is read, the lexical analyser searches the symbol table for an entry corresponding to the identifier. If there is no such entry, it then creates a new entry and returns a pointer along with the token. These additional values associated with individual tokens are called attributes.

1.b) ERROR RECOVERY IN THE LEXICAL ANALYSER

When an error is detected the lexical analyser should ideally continue with further lexical analysis without halting at the position where the error occurs. We achieved this by the simple method of discarding the offending line. The appropriate error message is printed.

List of lexemes along with their corresponding token values :

1	(	21	DIGIT
2	EVAR	22	ALPHA
3	X	23	.
4	Y	24	SIN
5	Z	25	SQR
6	UAO	26	COS
7	UOT	27	TAN
8	UIO	28	ARS
9	MIR	29	ARC
10	URT	30	ART
11	URP	31	ABS
12	SEF	32	NEG
13	RQO	33	INT
14	RPT	34	*
15	ERP	35	/
16	CLS	36	+
17	EPP	37	-
18	DIS	38	)
19	"	39	=
20	,	40	$

Data structures used :

1) Tokens are represented by nodes in a doubly linked list, each line being identified by a header node. These header nodes are also linked together. The nodes contain the following information.

* An integer to store the token type.
* Character variables to store the alpha numeric values.
* A flag to indicate whether the token is to be processed or not.
* A pointer to the symbol table location if the symbol is a E-variable.

2) The symbol table is also represented by a doubly linked list. The nodes of the symbol table contain information about the identifier name, and also the value of the identifier,depending on the identifier type.

The values of the identifiers are initialized to zero by default. A linear search routine is used to verify whether a symbol table entry has been made for a variable, when that variable is encountered by the lexical analyser.

A special function of the lexical analyser:

The lexical analyser performs a special task on encountering a call subroutine statement. The current location of the token pointer of the lexical analyser is pushed into a stack and the subroutine file is opened. Lexical analysis is carried out for that subroutine file as usual and after the lexical analysis of the entire subroutine file is done , the lexical analyser starts from the line following the call statement in the calling program.

A maximum of two nested subroutines are allowed in the parametric program. An error is indicated if there are more than two nested subroutine calls. The variables used in the subroutine can be accessed in the main program and vice versa. Hence there is no need for a separate symbol table. The token list for the subroutine is appended to the one for the calling program.

2. DESIGN OF THE SYNTAX ANALYSER :

We designed the syntax analyser using the LR parser.

WHAT IS A PARSER ?

A parser for a grammar "G" is a program that takes as input ,a string W, and produces as output,either a parse tree for W,if W is a sentence of G, or an error message,indicating that W is not a sentence of G.

LR PARSER :

The LR parser is an efficient bottom-up parser for context free grammars. This parser is called LR parser because it scans the input from left to right and constructs a right most derivation in reverse. This parser is attractive due to the following reasons:

1. LR parsers can be constructed to recognize virtually all programming language constructs for which context-free grammars can be written.
2. The LR parsing method is more general than operator-precedence or any other common shift-reduce techniques,yet it can be implemented with the same degree of efficiency as other methods. LR parsing also dominates the common forms of top-down parsing without backtracking.
3. LR parsers can detect syntactic errors as soon as it is possible to do so on a left-to-right scan of the input.

Figure 3 represents an LR parser. The parser has an input, a stack and a parsing table.

WORKING PRINCIPLE OF THE LR PARSER:

The input is read from left to right,one symbol at a time. The stack contains a string of the form S0 X1 S1 X2 S2 ..... Xm Sm ,where Sm is on top. Each Xi is a grammar symbol,and each Si is a symbol called a "state". Each state symbol summarises the information contained in the stack below it and is used to guide the shift-reduce decision. The parsing table consists of two parts, a parsing action function "ACTION" and goto function "GOTO".

The program driving the LR parser behaves as follows. It determines Sm, the state currently on top of the stack, and , ai , the current input symbol. It then consults ACTION [Sm,ai], which can have one of the four values:

(1) Shift S

(2) Reduce A -> β

(3) Accept

(4) Error

The function GOTO takes a stack and grammar symbol as arguments and produces a state. It is essentially the transition table of a deterministic finite automaton whose input symbols are the terminals and non-terminals of the grammar.

A configuration of an LR parser is a pair whose first component is the stack content and whose second component is the unexpended input:

(S0 X1 S1 X2 S2 ..... Xm Sm, ai ai+1 .....an$)

The next move of the parser is determined by reading ai,the current input symbol and Sm,the state on top of the stack, and then consulting the parsing action table entry ACTION [Sm,ai].

The configuration resulting after each of the four types of move are as follows:

(1) If ACTION[Sm,ai] = Shift S , the parser executes a shift move, entering the configuration

(S0 X1 S1 ..... Xm Sm ai S ,ai+1.....an $)

Here,the parser has shifted the current input symbol ai and the next state S = GOTO [Sm,ai] onto the stack : ai+1 becomes the current input symbol.

(2) If ACTION[Sm,ai] = Reduce A->β, then the parser executes a reduce move, entering the configuration

(S0 X1 S1 X2 S2.....Xm-r Sm-r a S, ai ai+1.....an$)

where S=GOTO[Sm-r,A] and r is the length of β,the right side of the production. Here the parser first popped 2r symbols of the stack(r state symbols ,r grammar symbols), exposing state Sm-r the parser then pushed both A , the left side of the production, and S,the entry for ACTION[Sm-r,a], onto the stack. The current input symbol is not changed in a reduce move. The sequence of the grammar symbols popped off the stack will always match the match the right side of the reducing production.

(3) If ACTION[Sm,ai] = "accept", parsing is completed.

(4) If ACTION[Sm,ai] = "error", the parser has discovered an error and call an error recovery routine.

The LR parsing algorithm is very simple. Initially the LR parser is in the configuration (S0 ,a1 a2 ..... an $) where S0 is a designated initial state and a1 a2 ..... an is the input string to be parsed. Then the parser executes moves until an accept or error action is encountered.

The canonical collection of LR(0) items:

WHY DO WE CONSTRUCT ITEMS ?

We define an LR(0) item of a grammar G to be a production of G, with a dot at some position of the right side.

We group items together into sets, which give rise to the states of an LR parser. The canonical LR(0) collection provides the basis for constructing a class of LR parsers called simple LR(SLR).

To construct the canonical LR(0) collection for a grammar, we need to define an augmented grammar and two functions CLOSURE and GOTO.

If G is a grammar with start symbol S,then G' , the augmented grammar for G, is G with a new start symbol S'and production S'->S. The purpose of this new starting production is to indicate to the parser, when it should stop parsing and announce acceptance of the input . This would occur when the parser was about to reduce by S'-> S.

CLOSURE:

If I is a set of items for grammar G ,then the set of items CLOSURE(I) is constructed from I by the rules:

1. Every item in I is in closure of I.
2. If A->α.Bβ is in closure of I and B->γ is a production,then add the item B->.γ to I,if it is not already there.

GOTO:

GOTO(I,X) is defined to be the closure of the set of all items (A->αX .β) such that (A->α.Xβ) is in I.

THE SET OF ITEMS CONSTRUCTION:

The canonical collection sets of LR(0) items for an augmented grammar G' is constructed by the following algorithm:

```
PROCEDURE ITEMS(G');
BEGIN
C:={CLOSURE ({S'->.S}) }
REPEAT
FOR each set of items I in C and each grammar symbol X such that
GOTO(I,X) is not empty and not in C DO
add GOTO(I,X) to C.
UNTIL no more sets of items can be added to C.
END.
```

CONSTRUCTION OF THE LR PARSING TABLE:

For constructing the SLR parsing table we require two functions: FIRST and FOLLOW associated with the grammar.

If α is any string of grammar symbols let FIRST(α) be the set of terminals that begin strings derived from α. If $\alpha \overset{*}{\rightarrow} \epsilon$ then ϵ is also in FIRST(α).

Algorithm to compute FIRST(X):

To compute FIRST(X) for all grammar symbols X,apply the following rules, until no more terminals or can be added to any FIRST set:

1. If X is a terminal, then FIRST(X) is {X}.
2. If X is a non-terminal and X->aα is a production, then add a to FIRST(X).If X->ϵ is a production, then add ϵ to FIRST(X).
3. If X->Y1 Y2 ..... Yk is a production,then for all i such that all of Y1 .....Yi-1 are non-terminals and FIRST(Yj) contains ϵ for j=1....i-1 (i.e. Y1 Y2 ..... Yi-1 $\stackrel{*}{=>}$ ϵ).Add every non-ϵ symbol in FIRST(Yi) to FIRST(X).If ϵ is in FIRST(Yj) for all j=1,2,3.....k,then add ϵ to FIRST(X).

   Now, we compute FIRST() for any string X1 X2.....Xn as follows.Add to FIRST(X1 X2.....Xn) all the non-ϵ symbols of FIRST(X1). Also add the non-ϵ symbols of FIRST(X1) and FIRST(X2) and so on. Finally, add ϵ to FIRST(X1 X2.....Xn) if ,for all i,FIRST(Xi) contains ϵ or if n=0.

Algorithm to compute FOLLOW(A):

To compute FOLLOW(A) for all non-terminals A,apply the following rules until nothing can be added to any FOLLOW set.

1. $ is in FOLLOW(S),where S is the start symbol.
2. If there is a production A->αBβ , $\beta \neq \epsilon$, then every thing in FIRST(β) but ϵ is in FOLLOW(B).
3. If there is a production A->αb,or a production A->αBβ where FIRST(β) contains ϵ (i.e $\beta \stackrel{*}{\rightarrow} \epsilon$),then everything in FOLLOW(A) is in FOLLOW(B).

CONSTRUCTION OF THE CONTEXT-FREE GRAMMAR:

For the syntactic specification of the NC parametric G-code programming language, we used a context-free grammar. The grammar consists of the following terminals, non terminals, start symbol and productions.

Terminals of the context-free grammar:

The terminals of the context-free grammar consists of the following tokens:

(,EVARIABLE,X,Y,Z,UAO,UOT,UIO,MIR,URT,URP,SCF,RQO,RPT,ERP,CLS, EPP, DIS,", , , DIGIT,ALPHABET, .,SIN,SQR,COS,TAN,ARS,ARC,ART,ABS, NEG,INT,*,/,+,-,) , =,$.

Non-terminals of the context-free grammar:

The non-terminals of the context-free grammar are the following symbols:

A,A1,B1,C1,D1,E1,F1,G1,H1,A2,B2,C2,D2,E2,F2,G2,H2,A3,A4,I,K,T1, T2,R,E,T,F.

The start symbol of the grammar is "S".

PRODUCTIONS OF THE CONTEXT-FREE GRAMMAR

S-> (A)/ EVAR=E / X E / Y E / Z E / .

A-> UAO A1 / UOT A2 / UIO B2 / MIR B1 / URT A3 / URP A3
/ SCF A4 / RQO A2 / RPT I / ERP / CLS K / EPP,K,K
/ DIS,"T2"

A1-> ,DIGIT T1 B1 / ,EVAR B1

B1-> ,X C1 / ,Y D1 / ,Z E1 / .

C1-> ,Y F1 / ,Z G1 / .

D1-> ,X F1 / ,Z H1 / .

E1-> ,Y H1 / ,X G1 / .

F1-> ,Z / .

G1-> ,Y / .

H1-> ,X / .

A2-> ,DIGIT T1 B2 / ,EVAR B2

B2-> ,X T1 C2 / ,Y T1 D2 / ,Z T1 E2 / .

C2-> ,Y T1 F2 / ,Z T1 G2 / .

D2-> ,X T1 F2 / ,Z T1 H2 / .

E2-> ,Y T1 H2 / ,X T1 G2 / .

F2-> ,Z T1 / .

G2-> ,Y T1 / .

H2-> ,X T1 / .

A3-> ,R / ,EVAR

A4-> ,R B1 / , EVAR B1 / .

I-> , DIGIT T1 / , EVAR

K-> , ALPHA T2

T1-> DIGIT T1 / .

T2-> ALPHA T2 / DIGIT T2 / .

R-> DIGIT R / .R / .

E-> E*T / E/T / T

T-> T+F / T-F / F

F-> -E / (E) / SIN (E) / SQR (E) / COS (E) / TAN(E) / ARS(E) /
ARC(E) / ART(E) / ABS(E) / INT(E) / NEG(E) / R / EVAR

The canonical collection of items of LR parser is given below:

I0 : S'->.S

S-> .(A)/ .EVAR=E / . X E / . Y E / . Z E / .
A-> . UAO A1 / . UOT A2 / . UIO B2 / . MIR B1 / . URT A3 / . URP A3 / . SCF A4 / . RQO A2 / . RPT I / . ERP / . CLS K / . EPP,K,K / . DIS,"T2"
A1-> . ,DIGIT T1 B1 / .,EVAR B1
B1-> . ,X C1 / . ,Y D1 / . ,Z E1 / .
C1-> .,Y F1 / . ,Z G1 / .
D1-> .,X F1 / .,Z H1 / .
E1-> . ,Y H1 / . ,X G1 / .
F1-> . ,Z / .
G1-> . ,Y / .
H1-> . ,X / .
A2-> . ,DIGIT T1 B2 / .,EVAR B2
B2-> .,X T1 C2 / .,Y T1 D2 / .,Z T1 E2 / .
C2-> .,Y T1 F2 / .,Z T1 G2 / .
D2-> .,X T1 F2 / .,Z T1 H2 / .
E2-> .,Y T1 H2 / .,X T1 G2 / .
F2-> .,Z T1 / .
G2-> .,Y T1 / .
H2-> .,X T1 / .
A3-> .,R / . ,EVAR
A4-> . ,R B1 / . , EVAR B1 / .
I-> . , DIGIT T1 / . , EVAR
K-> . , ALPHA T2
T1-> . DIGIT T1 / .
T2-> . ALPHA T2 / . DIGIT T2 / .
R-> . DIGIT R /. .R / .
E-> . E*T / . E/T / . T
T-> . T+F / .T-F / . F
F-> .-E / .(E) / . SIN (E) / .SQR (E) / .COS (E) / .TAN(E) / . ARS(E)/.ARC(E) / .ART(E) / . ABS(E) / . INT(E) / . NEG(E) /. R /.EVAR

I1 : GOTO(I0,S)

S'->S.

I2 : GOTO(I0,( )

S-> .(A)
A-> . UAO A1 / . UOT A2 / . UIO B2 / . MIR B1 / . URT A3/.URP A3 / . SCF A4 / . RQO A2 / . RPT I / . ERP / . CLS K /.EPP,K,K / . DIS,"T2"

F-> (.E)
R-> . DIGIT R /. .R / .
E-> . E*T / . E/T / . T
T-> . T+F / .T-F / . F
F-> .-E / .(E) / . SIN (E) / .SQR (E) / .COS (E) / .TAN(E) /
. ARS(E)/.ARC(E) / .ART(E) / . ABS(E) / . INT(E) / . NEG(E) /
. R / .EVAR

I3 : GOTO(I0,EVAR)
S-> EVAR. =E
F-> EVAR.

I4 : GOTO(I0,X)
S-> X . E
R-> . DIGIT R /. .R / .
E-> . E*T / . E/T / . T
T-> . T+F / .T-F / . F
F-> .-E / .(E) / . SIN (E) / .SQR (E) / .COS (E) / .TAN(E) /
.ARS(E)/.ARC(E) / .ART(E) / . ABS(E) / . INT(E) / . NEG(E)
/ . R / .EVAR

I5 : GOTO(I0,Y)
S-> Y . E
R-> . DIGIT R /. .R / .
E-> . E*T / . E/T / . T
T-> . T+F / .T-F / . F
F-> .-E / .(E) / . SIN (E) / .SQR (E) / .COS (E) / .TAN(E) /
.ARS(E)/.ARC(E) / .ART(E) / . ABS(E) / . INT(E) / . NEG(E)
/ . R / .EVAR

I6 : GOTO(I0,Z)
S-> Z . E
R-> . DIGIT R /. .R / .
E-> . E*T / . E/T / . T
T-> . T+F / .T-F / . F
F-> .-E / .(E) / . SIN (E) / .SQR (E) / .COS (E) / .TAN(E) /
. ARS(E)/.ARC(E) / .ART(E) / . ABS(E) / . INT(E) / . NEG(E)
/ . R / .EVAR

I7 : GOTO(I0,UAO)
A-> UAO .A1
A1-> . ,DIGIT T1 B1 / .,EVAR B1

I8 : GOTO(I0,UOT)

A-> UOT .A2
A2-> . ,DIGIT T1 B2 / .,EVAR B2

I9 : GOTO(I0,UIO)

A-> UIO .B2
B2-> .,X T1 C2 / .,Y T1 D2 / .,Z T1 E2 / .

I10 : GOTO(I0,MIR)

A-> MIR .B1
B1-> . ,X C1 / . ,Y D1 / . ,Z E1 / .

I11 : GOTO(I0,URT)

A-> URT .A3
A3-> .,R / . ,EVAR

I12 : GOTO(I0,URP)

A-> URP .A3
A3-> .,R / . ,EVAR

I13 : GOTO(I0,SCF)

A->SCF .A4
A4-> . ,R B1 / . , EVAR B1 / .

I14 : GOTO(I0,RQO)

A-> RQO .A2
A2-> . ,DIGIT T1 B2 / .,EVAR B2

I15 : GOTO(I0,RPT)

A-> RPT .I
I-> . , DIGIT T1 / . , EVAR

I16 : GOTO(I0,ERP)

A->ERP.

I17 : GOTO(I0,CLS)

A-> CLS .K
K-> . , ALPHA T2

I18 : GOTO(I0,EPP)

A-> EPP .,K,K

I19 : GOTO(I0,EPP)

A-> DIS .,"T2"

I20 : GOTO(I0,,)

A1-> ,.DIGIT T1 B1 / ,.EVAR B1
B1-> ,.X C1 / ,.Y D1 / ,.Z E1
C1-> ,.Y F1 / ,.Z G1
D1-> ,.X F1 / ,.Z H1
E1-> ,.Y H1 / ,.X G1
F1-> ,.Z
G1-> , .Y
H1-> , .X
A2-> , .DIGIT T1 B2 / , .EVAR B2

B2-> ,.X T1 C2 / ,.Y T1 D2 / ,.Z T1 E2
C2-> ,.Y T1 F2 / ,.Z T1 G2
D2-> ,.X T1 F2 / ,.Z T1 H2
E2-> ,.Y T1 H2 / ,.X T1 G2
F2-> ,.Z T1
G2-> ,.Y T1
H2-> ,.X T1
A3-> ,.R / ,.EVAR
A4-> ,.R B1 / , .EVAR B1
I-> ,. DIGIT T1 / ,. EVAR
K-> , .ALPHA T2
R-> .DIGIT R / . .R / .

I21 : GOTO (I0, DIGIT)

T1-> DIGIT . T1
T1-> . DIGIT T1 / .
T2-> DIGIT . T2
T2-> . ALPHA T2 / . DIGIT T2 / .
R-> DIGIT . R
R-> . DIGIT R / . .R / .

I22 : GOTO (I0, ALPHA)

T2-> ALPHA . T2
T2-> . ALPHA T2 / . DIGIT T2 / .

I23 : GOTO (I0,.)

R-> . .R
R-> . DIGIT R / . .R / .

I24 : GOTO (I0,E)

E-> E . * T / E . IT

I25 : GOTO (I0,T)

E-> T .
T-> T . + F / T . - F

I26 : GOTO (10,F)

T-> F .

I27 : GOTO (I0,-)

F-> . E
E-> . E * T / . E / T / . T
T-> . T + F / . T - F / . F
F-> . - E / . (E) / . TRIG (E) / . R / . ERAR
R-> . DIGIT R / . .. / .

I28 : GOTO (I0,SIN)

F-> SIN (E)

I29 : GOTO (10,SQR)

F-> SQR . (E)

I30 : GOTO (I0,COS)
F-> COS . (E)
I31 : GOTO (I0,TAN)
F-> TAN . (E)
I32 : GOTO (I0,ARS)
F-> ARS . (E)
I33 : GOTO (I0,ARC)
F-> ARC . (E)
I34 : GOTO (I0,ART)
F-> ART . (E)
I35 : GOTO (I0,ABS)
F-> ABS . (E)
I36 : GOTO (I0,INT)
F-> INT .(E)
I37 : GOTO (I0,NEG)
F-> NEG . (E)
I38 : GOTO (I0,.R)
F-> R .
I39 : GOTO (I2,A)
S-> A .
GOTO (I2,UAO) = I7
GOTO (I2,UOT) = I8
GOTO (I2,UIO) = I9
GOTO (I2,MIR) = I10
GOTO (I2,URT) = I11
GOTO (I2,URP) = I12
GOTO (I2,SCF) = I13
GOTO (I2,RQO) = I14
GOTO (I2,RPT) = I15
GOTO (I2,ERP) = I16
GOTO (I2,CLS) = I17
GOTO (I2,EPP) = I18
GOTO (I2,DIS) = I19
I40 : GOTO (I2,E)
F-> (E .)
E-> E . * T / E . / T
GOTO (I2,DIGIT) = I21
GOTO (I2, .) = I23
GOTO (I2,T) = I25
GOTO (I2,F) = I26
GOTO (I2,-) = I27
GOTO (I2,SIN) = I28

GOTO (I2,SQR) = I29
GOTO (I2,COS) = I30
GOTO (I2,TAN) = I31
GOTO (I2,ARS) = I32
GOTO (I2,ARC) = I33
GOTO (I2,ART) = I34
GOTO (I2,ABS) = I35
GOTO (I2,INT) = I36
GOTO (I2,NEG) = I37
GOTO (I2,R) = I38

I41 **: GOTO (I2,EVAR)**

F-> EVAR .

I42 **: GOTO (I3,=)**

S-> EVAR = . E
E-> . E * T / . E / T / . T
T-> . T + F / . T - F / . F
F-> . E / . (E) / . TRIG (E) / . R / . EVAR
R-> . DIGIT R / . .R / .

I43 **: GOTO (I4,E)**

S-> X E .
E-> E .* T / E. / T
GOTO (I4,T) = I25
GOTO (I4,F) = I26
GOTO (I4,-) = I27
GOTO (I4,SIN) = I28
GOTO (I4,SQR) = I29
GOTO (I4,COS) = I30
GOTO (I4,TAN) = I31
GOTO (I4,ARS) = I32
GOTO (I4,ARC) = I33
GOTO (I4,ART) = I34
GOTO (I4,ABS) = I35
GOTO (I4,INT) = I36
GOTO (I4,NEG) = I37
GOTO (I4,R) = I38
GOTO (I4,EVAR) = I41
GOTO (I4,DIGIT) = I78
GOTO (I4,.) = I23

I44 **: GOTO (I4,()**

F-> (.E)
E-> .E * T / .E /T /.T
T-> .T + F /.T - F /.F

F-> .- E / .(E) / .trig (E) / .R /.EVAR
GOTO (I44,E) = I40
GOTO (I44,T) = I25
GOTO (I44,F) = I26
GOTO (I44,-) = I27
GOTO (I44,SIN) = I28
GOTO (I44,SQR) = I29
GOTO (I44,COS) = I30
GOTO (I44,TAN) = I31
GOTO (I44,ARS) = I32
GOTO (I44,ARC) = I33
GOTO (I44,ART) = I34
GOTO (I44,ABS) = I35
GOTO (I44,INT) = I36
GOTO (I44,NEG) = I37
GOTO (I44,R) = I38

I45 **: GOTO (I5,E)**

S-> Y E.
E-> E .* T / E ./ T
GOTO (I5,T) = I25
GOTO (I5,F) = I26
GOTO (I5,-) = I27
GOTO (I5,SIN) = I28
GOTO (I5,SQR) = I29
GOTO (I5,COS) = I30
GOTO (I5,TAN) = I31
GOTO (I5,ARS) = I32
GOTO (I5,ARC) = I33
GOTO (I5,ART) = I34
GOTO (I5,ABS) = I35
GOTO (I5,INT) = I36
GOTO (I5,NEG) = I37
GOTO (I5,R) = I38
GOTO (I5,EVAR) = I41
GOTO (I5,DIGIT) = I78
GOTO (I5,.) = I23

I46 **: GOTO (I6,E)**

S-> Z E.
E-> E .* T / E./T
GOTO (I6,T) = I25
GOTO (I6,F) = I26
GOTO (I6,-) = I27

GOTO (I6,SIN) = I28
GOTO (I6,SQR) = I29
GOTO (I6,COS) = I30
GOTO (I6,TAN) = I31
GOTO (I6,ARS) = I32
GOTO (I6,ARC) = I33
GOTO (I6,ART) = I34
GOTO (I6,ABS) = I35
GOTO (I6,INT) = I36
GOTO (I6,NEG) = I37
GOTO (I6,R) = I38
GOTO (I6,EVAR) = I41
GOTO (I6,DIGIT) = I78
GOTO (I6,.) = I23

I47 : GOTO (I7,A1)

A-> UAO A1.

I48 : GOTO (I7, ,)

A1-> , .DIGIT T1 B1 / ,.EVAR B1

I49 : GOTO (I8,A2)

A-> UOT A2.

I50 : GOTO (I8, ,)

A2-> .DIGIT T1 B2 / ,.EVAR B2

I51 : GOTO (I9,B2)

A-> UIO B2.

I52 : GOTO (I9, ,)

B2-> , .X T1 C2 / , .Y T1 D2 / , .Z T1 E2

I53 : GOTO (I10,B1)

A-> MIR B1.

I54 : GOTO (I10, ,)

B1-> , .X C1 / , .Y D1 / , .Z E1

I55 : GOTO (I11,A3)

A-> URT A3.

I56 : GOTO (I11, ,)

A3-> , .R / , .EVAR

R-> .DIGIT R / .R / .

I57 : GOTO (I12,A3)

A-> URP A3.

GOTO (I12, ,) = I56

I58 : GOTO (I13,A4)

A-> SCF A4.

I59 : GOTO (I13, ,)

A4-> , .R B1 / , .EVAR B1

R-> .DIGIT R / . .R / .

I60 : GOTO (I14,A2)

A-> RQO A2.

GOTO (I14, ,) = I49

I61 : GOTO (I15,I)

A-> RPT I.

I62 : GOTO (I15, ,)

I-> , .DIGIT T1 / , .EVAR

I63 : GOTO (I17,K)

A-> CLS K.

I64 : GOTO (I17, ,)

K-> ,.ALPHA T2

I65 : GOTO (I18, ,)

A-> EPP,.K,K

K-> .,ALPHA T2

I66 : GOTO (I19, ,)

A-> DIS , ."T2"

I67 : GOTO (I20,DIGIT)

A1-> , DIGIT .T1 B1

A2-> , DIGIT .T1 B2

T1-> .DIGIT T1 / .

R-> DIGIT .R

R-> .DIGIT R / . .R / .

I-> ,DIGIT .T1

I68 : GOTO (I20,EVAR)

A1-> ,EVAR .B1

B1-> ., X C1 / ., Y D1 / ., Z E1 / .

A2-> ,EVAR .B2

B2-> ., X T1 C2 / . ,Y T1 D2 / .,Z T1 E2 / .

A3-> ,EVAR.

I-> ,EVAR.

I69 : GOTO (I20,X)

B1-> , X .C1

C1-> .,Y F1 / .,Z G1 / .

D1-> , X .F1

F1-> ., Z / .

B2-> , X .T1 C2

T1-> .DIGIT T1 / .

D2-> , X .T1 F2

E2-> , X .T1 G2

H2-> , X .T1

H-> , X.

E1-> , X .G1
G1-> ., Y / .

I70 : GOTO(I20,Y)

B1-> , Y .D1
D1-> ., X F1 / ., Z H1 / .
C1-> , Y .F1
F1-> ., Z / .
E1-> , Y .H1
H1-> ., X / .
G1-> , Y.
B2-> , Y .T1 D2
T1-> .DIGIT T1 / .
C2-> ,Y .T1 F2
E2-> , Y .T1 H2
G2-> , Y .T1

I71 : GOTO(I20,Z)

B1-> , Z . E1
E1-> ., YH1 / ., X G1 / .
C1-> , Z .G1
G1-> ., Y / .
D1-> , Z .H1
H1-> ,. ,X / .
F1-> , Z.
B2-> , Z. T1 E2
T1-> .DIGIT T1 / .
C2-> , Z .T1 G2
D2-> , Z .T1 H2
F2-> , Z .T1

I72 : GOTO (I20,R)

A3-> , R.
A4-> , R .B1
B1-> ., X C1 / ., Y D1 / ., Z E1 / .
GOTO (I20, .) = I23
GOTO (I20, ALPHA) = I22

I73 : GOTO (I21,T1)

T1-> DIGIT T1.

I74 : GOTO (I21,T2)

T2-> ALPHA T2.
GOTO (I21,ALPHA) = I22
GOTO (I21, DIGIT) = I21
GOTO (I21,.) = I23

I75 : GOTO (I21,R)

R-> DIGIT R.
GOTO (I22,T2) = I73
GOTO (I22,ALPHA) = I22

I76 : GOTO (I22, DIGIT)
T2-> DIGIT .D2
T2-> .ALPHA T2 / .DIGIT T2 / .

I77 : GOTO (I23,R)
R-> .R

I78 : GOTO (I23,DIGIT)
R-> DIGIT .R
R-> .DIGIT R / . .R / .

I79 : GOTO (I24,*)
E-> E * .T
T-> .T + F / .T - F / .F
F-> .- E / .(E) /.trig(E) / .R / .EVAR
R-> .DIGIT R / . .R / .

I80 : GOTO (I24,/)
E-> E / .T
T-> .T + F / .T - F / .F
F-> .- E / .(E) / .trig (E) / .R / .EVAR
R-> .DIGIT R / . .R / .

I81 : GOTO (I25,+)
T-> T + .F
F-> .- E / .(E) / .trig (E) / .R / .EVAR
R-> .DIGIT R / . .R / .

I82 : GOTO (I25,-)
T-> T - .F
F-> .- E / .(E) / .trig (E) / .R / .EVAR
R-> .DIGIT R / . .R / .

I83 : GOTO (I27,E)
F-> - E .
E-> E .* T / E. / T
GOTO (I27,T) = I25
GOTO (I27,F) = I26
GOTO (I27,-) = I27
GOTO (I27,SIN) = I28
GOTO (I27,SQR) = I29
GOTO (I27,COS) = I30
GOTO (I27,TAN) = I31
GOTO (I27,ARS) = I32
GOTO (I27,ARC) = I33
GOTO (I27,ART) = I34

GOTO (I27,ABS) = I35
GOTO (I27,INT) = I36
GOTO (I27,NEG) = I37
GOTO (I27,R) = I38
GOTO (I27,EVAR) = I41
GOTO (I27,DIGIT) = I78
GOTO (I27,.) = I23

I84 : GOTO (I28,( )

F-> SIN (.E)
E-> .E * T / .E / T / .T
T-> .T + F / .T - F / .F
F-> .- E / .(E) / .trig (E) / .R / .EVAR
R-> .DIGIT R / . .R / .

I85 : GOTO (I29,( )

F-> SQR (.E)
E-> .E * T / .E / T / .T
T-> .T + F / .T - F / .F
F-> .- E / .(E) / .trig (E) / .R / .EVAR
R-> .DIGIT R / . .R / .

I86 : GOTO (I30,( )

F-> COS (.E)
E-> .E * T / .E / T / .T
T-> .T + F / .T - F / .F
F-> .- E / .(E) / .trig (E) / .R / .EVAR
R-> .DIGIT R / . .R / .

I87 : GOTO (I31,( )

F-> TAN (.E)
E-> .E * T / .E / T / .T
T-> .T + F / .T - F / .F
F-> .- E / .(E) / .trig (E) / .R / .EVAR
R-> .DIGIT R / . .R / .

I88 : GOTO (I32,( )

F-> ARS (.E)
E-> .E * T / .E / T / .T
T-> .T + F / .T - F / .F
F-> .- E / .(E) / .trig (E) / .R / .EVAR
R-> .DIGIT R / . .R / .

I89 : GOTO (I33,( )

F-> ARC (.E)
E-> .E * T / .E / T / .T
T-> .T + F / .T - F / .F
F-> .- E / .(E) / .trig (E) / .R / .EVAR

R-> .DIGIT R / . .R / .

I90 : GOTO (I34,( )

F-> ART (.E)
E-> .E * T / .E / T / .T
T-> .T + F / .T - F / .F
F-> .- E / .(E) / .trig (E) / .R / .EVAR
R-> .DIGIT R / . .R / .

I91 : GOTO (I35,( )

F-> ABS (.E)
E-> .E * T / .E / T / .T
T-> .T + F / .T - F / .F
F-> .- E / .(E) / .trig (E) / .R / .EVAR
R-> .DIGIT R / . .R / .

I92 : GOTO (I36,( )

F-> INT (.E)
E-> .E * T / .E / T / .T
T-> .T + F / .T - F / .F
F-> .- E / .(E) / .trig (E) / .R / .EVAR
R-> .DIGIT R / . .R / .

I93 : GOTO (I37,( )

F-> NEG (.E)
E-> .E * T / .E / T / .T
T-> .T + F / .T - F / .F
F-> .- E / .(E) / .trig (E) / .R / .EVAR
R-> .DIGIT R / . .R / .

I94 : GOTO (I39,)

S-> (A).

I95 : GOTO (I40,)

F-> (E) .
GOTO (I40, *) = I78
GOTO (I40, /) = I79

I96 : GOTO (I42,E)

S-> EVAR = E.
E-> E .* T / E ./ T
GOTO (I42,T) = I25
GOTO (I42,F) = I26
GOTO (I42,-) = I27
GOTO (I42,SIN) = I28
GOTO (I42,SQR) = I29
GOTO (I42,COS) = I30
GOTO (I42,TAN) = I31
GOTO (I42,ARS) = I32

GOTO (I42,ARC) = I33
GOTO (I42,ART) = I34
GOTO (I42,ABS) = I35
GOTO (I42,INT) = I36
GOTO (I42,NEG) = I37
GOTO (I42,R) = I38
GOTO (I42,EVAR) = I41
GOTO (I42,DIGIT) = I78
GOTO (I42,.) = I23
GOTO (I43,*) = I78
GOTO (I43,/) = I79
GOTO (I45,*) = I78
GOTO (I45,/) = I79
GOTO (I46,*) = I78
GOTO (I46,/) = I79

I97 : GOTO (I47,DIGIT)
A1-> , DIGIT .T1 B1
T1-> .DIGIT T1 / .

I98 : GOTO (I48,EVAR)
A1-> , EVAR .B1
B1-> ., X C1 /., Y D1 / ., Z E1 / .

I99 : GOTO (I50,DIGIT)
A2-> , DIGIT . T1 B2
T1-> . DIGIT T1 / .

I100 : GOTO (I50,EVAR)
A2-> , EVAR .B2
B2-> ., X T1 C2 / ., Y T1 D2 / ., Z T1 E2 / .

I101 : GOTO (I52,X)
B2-> ,X .T1 C2
T1-> .DIGIT T1 / .

I102 : GOTO (I52,Y)
B2-> , Y .T1 D2
T1-> .DIGIT T1 / .

I103 : GOTO (I52,Z)
B2-> , Z .T1 E2
T1-> .DIGIT T1 / .

I104 : GOTO (I54,X)
B1-> .X C1
C1-> ., YF1 / ., Z G1 / .

I105 : GOTO (I54,Y)
B1-> , Y .D1
D1-> ., X F1 / ., Z H1 / .

I106 : GOTO (I54,Z)
B1-> , Z .E1
E1-> ., Y H1 / ., X G1 / .
I107 : GOTO (I56,R)
A3-> , R.
I108 : GOTO (I56,EVAR)
A3-> , EVAR.
GOTO (I56,DIGIT) = I77
GOTO (I56,.) = I23
I109 : GOTO (I59,R)
A4-> , R .B1
B1-> ., X C1 / ., Y D1 / ., Z E1 / .
I110 : GOTO (I59,EVAR)
A4-> , EVAR .B1
B1-> ., X C1 / ., Y D1 / ., Z E1 / .
GOTO(I59,DIGIT) = I4
GOTO(I59,.) = I23
I111 : GOTO (I62,DIGIT)
I-> , DIGIT .T1
T1-> .DIGIT T1 / .
I112 : GOTO (I62,EVAR)
I-> , EVAR.
I113 : GOTO (I64,ALPHA)
K-> , ALPHA .T2
T2-> .ALPHA T2 / .DIGIT T2 / .
I114 : GOTO (I65,K)
B-> EPP , K ., K
GOTO (I64,)) = I63
I115 : GOTO (I66,")
S-> DIS , " .T2 "
T2-> .ALPHA T2 / . DIGIT T2 / .
I116 : GOTO (I67,T1)
A1-> , DIGIT T1 .B1
I-> , DIGIT T1.
B1-> .,XC1 / ., Y D1 / ., Z E1 / .
A2-> , DIGIT T1 .B2
B2-> ., X T1 C2 / ., Y T1 D2 / ., Z T1 E2
I117 : GOTO (I67,DIGIT)
T1-> DIGIT .T1
T1-> .DIGIT T1 / .
R-> .DIGIT R
R-> .DIGIT R / . .R / .

GOTO (I67,.) = I23
GOTO (I67,( ) = I74

I118 : GOTO (I68,B1)
A1-> , EVAR B1.

I119 : GOTO (I68 , B2)
A2-> , EVAR B2.

I120 : GOTO (I68, ,)
B1-> , .X C1 / , .Y D1 / , .Z E1
B2-> , .X T1 C2 / , .Y T1 D2 / , .Z T1 E2

I121 : GOTO (I69,C1)
B1-> , X C1.

I122 : GOTO (I69,F1)
D1-> , X F1.

I123 : GOTO (I69, ,)
C1-> , .Y F1 / , .Z G1 / .
F1-> , .Z
G1-> , .Y

I124 : GOTO (I69,T1)
B2-> , X T1 .C2
C2-> ., Y T1 F2 / ., Z T1 G2 / .
D2-> , X T1 .F2
E2-> , X T1 .G2
F2-> ., Z T1 / .
G2-> ., Y T1 / .

I125 : GOTO (I69,DIGIT)
T1-> DIGIT .T1
T1-> .DIGIT T1 / .

I126 : GOTO (I69,G1)
E1-> , X G1.

I127 : GOTO (I70,D1)
B1->, Y D1.

I128 : GOTO (I70,F1)
C1-> , Y F1.

I129 : GOTO (I70,H1)
E1-> , Y H1.

I130 : GOTO (I70 , ,)
D1-> , .X F1 / , .Z H1
F1-> , .Z
H1-> , .X

I131 : GOTO (I70,T1)
B2-> , Y T1 .D2
D2-> ., X T1 F2 / ., Z T1 H2 / .

C2-> , Y T1 .F2
F2-> ., Z T1 / .
E2-> , Y T1 .H2
H2-> ., X T1 / .
G2-> , Y T1
GOTO (I70,DIGIT) = I125

I132 : GOTO (I71,E1)
B1-> , Z E1

I133 : GOTO (I71, G1)
G1-> , Z G1.

I134 : GOTO (I71 , H1)
D1-> , Z H1.

I135 : GOTO (I71, ,)
C1-> , .Y H1 / , .XG1
G1-> , .Y
H1-> , .X

I136 : GOTO (I71,T1)
B2-> , Z T1 .E2
C2-> , Z T1 .G2
D2-> , Z T1 .H1
F2-> , Z T1.
E2-> ., Y T1 H2 / ., X T1 G2 / .
G2-> ., Y T1 / .
H2-> ., X T1 / .
GOTO (I71,DIGIT) = I125

I137 : GOTO (I72,B1)
A4-> , R B1.
GOTO (I71, ,) = I54

I138 : GOTO (I76,T2)
T2-> DIGIT T2.
GOTO (I76,ALPHA) = I22
GOTO (I76,DIGIT) = I76
GOTO (I78,R) = I75
GOTO (I78,DIGIT) = I78
GOTO (I78, .) = I23

I139 : GOTO (I79,T)
E-> E * T.
T-> T .+ F / T .- F
GOTO (I79,-) = I27
GOTO (I79,SIN) = I28
GOTO (I79,SQR) = I29
GOTO (I79,COS) = I30

GOTO (I79,TAN) = I31
GOTO (I79,ARS) = I32
GOTO (I79,ARC) = I33
GOTO (I79,ART) = I34
GOTO (I79,ABS) = I35
GOTO (I79,INT) = I36
GOTO (I79,NEG) = I37
GOTO (I79,R) = I38
GOTO (I79,EVAR) = I41
GOTO (I79,DIGIT) = I78
GOTO (I79,.) = I23
GOTO (I79,F) = I26

I140 : GOTO (I80,T)

E-> E / T.
T-> T .+ F / T .- F
GOTO (I80,-) = I27
GOTO (I80,SIN) = I28
GOTO (I80,SQR) = I29
GOTO (I80,COS) = I30
GOTO (I80,TAN) = I31
GOTO (I80,ARS) = I32
GOTO (I80,ARC) = I33
GOTO (I80,ART) = I34
GOTO (I80,ABS) = I35
GOTO (I80,INT) = I36
GOTO (I80,NEG) = I37
GOTO (I80,R) = I38
GOTO (I80,EVAR) = I41
GOTO (I80,DIGIT) = I78
GOTO (I80,.) = I23
GOTO (I80,F) = I26

I141 : GOTO (I81,F)

T-> T + F.
GOTO (I81,-) = I27
GOTO (I81,SIN) = I28
GOTO (I81,SQR) = I29
GOTO (I81,COS) = I30
GOTO (I81,TAN) = I31
GOTO (I81,ARS) = I32
GOTO (I81,ARC) = I33
GOTO (I81,ART) = I34
GOTO (I81,ABS) = I35

GOTO (I81,INT) = I36
GOTO (I81,NEG) = I37
GOTO (I81,R) = I38
GOTO (I81,EVAR) = I41
GOTO (I81,DIGIT) = I78
GOTO (I81,.) = I23

I142 : GOTO (I82,F)

T-> T - F.
GOTO (I82,-) = I27
GOTO (I82,SIN) = I28
GOTO (I82,SQR) = I29
GOTO (I82,COS) = I30
GOTO (I82,TAN) = I31
GOTO (I82,ARS) = I32
GOTO (I82,ARC) = I33
GOTO (I82,ART) = I34
GOTO (I82,ABS) = I35
GOTO (I82,INT) = I36
GOTO (I82,NEG) = I37
GOTO (I82,R) = I38
GOTO (I82,EVAR) = I41
GOTO (I82,DIGIT) = I78
GOTO (I82,.) = I23
GOTO (I83,*) = I79
GOTO (I83,/) = I80
GOTO (I83,DIGIT) = I78
GOTO (I83,.) = I23

I143 : GOTO (I84,E)

F-> SIN (E.)
E-> E .* T / E ./ T
GOTO (I84,-) = I27
GOTO (I84,SIN) = I28
GOTO (I84,SQR) = I29
GOTO (I84,COS) = I30
GOTO (I84,TAN) = I31
GOTO (I84,ARS) = I32
GOTO (I84,ARC) = I33
GOTO (I84,ART) = I34
GOTO (I84,ABS) = I35
GOTO (I84,INT) = I36
GOTO (I84,NEG) = I37
GOTO (I84,R) = I38

GOTO (I84,EVAR) = I41
GOTO (I84,DIGIT) = I78
GOTO (I84,.) = I23
GOTO (I84,F) = I26
GOTO (I84,T) = I25

I145 : GOTO (I85,E)

F-> SQR (E .)
E-> E.* T / E ./ T
GOTO (I85,-) = I27
GOTO (I85,SIN) = I28
GOTO (I85,SQR) = I29
GOTO (I85,COS) = I30
GOTO (I85,TAN) = I31
GOTO (I85,ARS) = I32
GOTO (I85,ARC) = I33
GOTO (I85,ART) = I34
GOTO (I85,ABS) = I35
GOTO (I85,INT) = I36
GOTO (I85,NEG) = I37
GOTO (I85,R) = I38
GOTO (I85,EVAR) = I41
GOTO (I85,DIGIT) = I78
GOTO (I85,.) = I23
GOTO (I85,F) = I26
GOTO (I85,T) = I25

I146 : GOTO (I86,E)

F-> COS (E .)
E-> E .* T / E ./ T
GOTO (I86,-) = I27
GOTO (I86,SIN) = I28
GOTO (I86,SQR) = I29
GOTO (I86,COS) = I30
GOTO (I86,TAN) = I31
GOTO (I86,ARS) = I32
GOTO (I86,ARC) = I33
GOTO (I86,ART) = I34
GOTO (I86,ABS) = I35
GOTO (I86,INT) = I36
GOTO (I86,NEG) = I37
GOTO (I86,R) = I38
GOTO (I86,EVAR) = I41
GOTO (I86,DIGIT) = I78

GOTO (I86,.) = I23
GOTO (I86,F) = I26
GOTO (I86,T) = I25

I147 : GOTO (I87 , E)

F-> TAN (E.)
E-> E .* T / E ./ T
GOTO (I87,-) = I27
GOTO (I87,SIN) = I28
GOTO (I87,SQR) = I29
GOTO (I87,COS) = I30
GOTO (I87,TAN) = I31
GOTO (I87,ARS) = I32
GOTO (I87,ARC) = I33
GOTO (I87,ART) = I34
GOTO (I87,ABS) = I35
GOTO (I87,INT) = I36
GOTO (I87,NEG) = I37
GOTO (I87,R) = I38
GOTO (I87,EVAR) = I41
GOTO (I87,DIGIT) = I78
GOTO (I87,.) = I23
GOTO (I87,F) = I26
GOTO (I87,T) = I25

I148 : GOTO (I88,E)

F-> ARS (E.)
E-> E .* T / E./T

GOTO (I88,-) = I27
GOTO (I88,SIN) = I28
GOTO (I88,SQR) = I29
GOTO (I88,COS) = I30
GOTO (I88,TAN) = I31
GOTO (I88,ARS) = I32
GOTO (I88,ARC) = I33
GOTO (I88,ART) = I34
GOTO (I88,ABS) = I35
GOTO (I88,INT) = I36
GOTO (I88,NEG) = I37
GOTO (I88,R) = I38
GOTO (I88,EVAR) = I41
GOTO (I88,DIGIT) = I78
GOTO (I88,.) = I23

GOTO (I88,F) = I26
GOTO (I88,T) = I25

I149 : GOTO (I89,E)

F-> ARC (E .)
E-> E .* T / E ./ T
GOTO (I89,-) = I27
GOTO (I89,SIN) = I28
GOTO (I89,SQR) = I29
GOTO (I89,COS) = I30
GOTO (I89,TAN) = I31
GOTO (I89,ARS) = I32
GOTO (I89,ARC) = I33
GOTO (I89,ART) = I34
GOTO (I89,ABS) = I35
GOTO (I89,INT) = I36
GOTO (I89,NEG) = I37
GOTO (I89,R) = I38
GOTO (I89,EVAR) = I41
GOTO (I89,DIGIT) = I78
GOTO (I89,.) = I23
GOTO (I89,F) = I26
GOTO (I89,T) = I25

I150 : GOTO (I90,E)

F-> ART (E .)
E-> E .* T / E ./ T
GOTO (I90,-) = I27
GOTO (I90,SIN) = I28
GOTO (I90,SQR) = I29
GOTO (I90,COS) = I30
GOTO (I90,TAN) = I31
GOTO (I90,ARS) = I32
GOTO (I90,ARC) = I33
GOTO (I90,ART) = I34
GOTO (I90,ABS) = I35
GOTO (I90,INT) = I36
GOTO (I90,NEG) = I37
GOTO (I90,R) = I38
GOTO (I90,EVAR) = I41
GOTO (I90,DIGIT) = I78
GOTO (I90,.) = I23
GOTO (I90,F) = I26
GOTO (I90,T) = I25

I151 : GOTO (I91,E)

F-> ABS (E .)
E-> E .* T / E ./ T
GOTO (I91,-) = I27
GOTO (I91,SIN) = I28
GOTO (I91,SQR) = I29
GOTO (I91,COS) = I30
GOTO (I91,TAN) = I31
GOTO (I91,ARS) = I32
GOTO (I91,ARC) = I33
GOTO (I91,ART) = I34
GOTO (I91,ABS) = I35
GOTO (I91,INT) = I36
GOTO (I91,NEG) = I37
GOTO (I91,R) = I38
GOTO (I91,EVAR) = I41
GOTO (I91,DIGIT) = I78
GOTO (I91,.) = I23
GOTO (I91,F) = I26
GOTO (I91,T) = I25

I152 : GOTO (I92,E)

F-> INT (E .)
E-> E .* T / E ./ T
GOTO (I92,-) = I27
GOTO (I92,SIN) = I28
GOTO (I92,SQR) = I29
GOTO (I92,COS) = I30
GOTO (I92,TAN) = I31
GOTO (I92,ARS) = I32
GOTO (I92,ARC) = I33
GOTO (I92,ART) = I34
GOTO (I92,ABS) = I35
GOTO (I92,INT) = I36
GOTO (I92,NEG) = I37
GOTO (I92,R) = I38
GOTO (I92,EVAR) = I41
GOTO (I92,DIGIT) = I78
GOTO (I92,.) = I23
GOTO (I92,F) = I26
GOTO (I92,T) = I25

I53 : GOTO (I93, E)

F-> NEG (E .)

E-> E .* T / E ./ T
GOTO (I93,-) = I27
GOTO (I93,SIN) = I28
GOTO (I93,SQR) = I29
GOTO (I93,COS) = I30
GOTO (I93,TAN) = I31
GOTO (I93,ARS) = I32
GOTO (I93,ARC) = I33
GOTO (I93,ART) = I34
GOTO (I93,ABS) = I35
GOTO (I93,INT) = I36
GOTO (I93,NEG) = I37
GOTO (I93,R) = I38
GOTO (I93,EVAR) = I41
GOTO (I93,DIGIT) = I78
GOTO (I93,.) = I23
GOTO (I93,F) = I26
GOTO (I93,T) = I25
GOTO (I96,*) = I78
GOTO (I96,/) = I79

I154 : GOTO (I97,T1)

A1-> , DIGIT T1 .B1
B1-> ., X C1 / ., Y D1 / ., Z E1 / .
GOTO (I97,DIGIT) = I125
GOTO (I98,B1) = I118
GOTO (I98, ,) = I54

I155 : GOTO (I99,T1)

A1-> , DIGIT T1 .B2
B1-> ., X C2 / ., Y T1 D2 / ., Z T1 E2 / .
GOTO (I99,DIGIT) = I125
GOTO (I100,B2) = I119
GOTO (I00, ,) = I52

I156 : GOTO (I101,T1)

B2-> , X T1 .C2
C2-> ., Y T1 F2 / ., Z T1 G2 / .
GOTO (I101, DIGIT) = I125

I157 : GOTO (I102,T1)

B2-> , Y T1 .D2
D2-> ., X T1 F2 / ., Z T1 H2 / .
GOTO (I102, DIGIT) = I125

I158 : GOTO (I103,T1)

B2-> , Z T1 .E2

E2-> ., Y T1 H2 / ., X T1 G2 / .
GOTO (I103, DIGIT) = I125
GOTO (I103, C1) = I121

I159 : GOTO (I104, ,)

C1-> , .Y F1/ , .Z G1
GOTO (I105, D1) = I127

I160 : GOTO (I105, ,)

D1-> , .X F1/ , .Z H1
GOTO (I106, E1) = I132

I161 : GOTO (I106, ,)

E1-> , .Y H1/ , .X G1
GOTO (I109, B1) = I137
GOTO (I109, ,) = I54

I162 : GOTO (I110,B1)

A4-> , EVAR B1 .
GOTO (I110, ,) = I54

I163 : GOTO (I111,T1)

I-> , DIGIT T1 .
GOTO (I111, DIGIT) = I125

I164 : GOTO (I113, T2)

K-> , ALPHA T2 .
GOTO (I113, ALPHA) = I22
GOTO (I113, DIGIT) = I76

I165 : GOTO (I114, ,)

S-> EPP , K, .K
K-> . , ALPHA T2

I166 : GOTO (I115,T2)

S-> DIS,"T2."
GOTO (I115, ALPHA) = I22
GOTO (I115, DIGIT) = I76

I167 : GOTO (I116,B1)

A1-> , DIGIT T1 B1.

I168 : GOTO (I116,B2)

A2-> , DIGIT T1 B2.
GOTO (I116, ,) = I120
GOTO (I117, T1) = I72
GOTO (I117, R) = I74
GOTO (I117, DIGIT) = I117
FOTO (I117, .) = I 23

I169 : GOTO (S120,X)

B1-> , X .C1
B2-> , X .T1 C2

C1-> ., Y F1 / ., Z G1 /.
T1-> . DIGIT T1 / .

I170 : GOTO (S120,Y)

B1-> , Y .D1
B2-> , Y .T1 D2
D1-> ., X F1 / ., Z H1 / .
T1-> . DIGIT T1 / .

I171 : GOTO (S120,Z)

B1-> , Z .E1
B2-> , Z .T1 E2
E1-> ., Y H1 / ., Z G1 / .
T1-> . DIGIT T1 / .

I172 : GOTO (I123,Y)

C1-> , Y .F1
G1-> , Y.
F1-> ., Z / .

I173 : GOTO (I123,Z)

C1-> , Z .G1
F1-> , Z.
G1-> ., Y / .

I174 : GOTO (I124,C2)

S2-> , X T1 C2 .

I175 : GOTO (I124, ,)

C2-> / .Y T1 F2 / , .Z T1 G2
F2-> / . Z T1 /
G2 -> , . Y T1

I176 : GOTO (I124,F2)

D2-> , X T1 F2 .

I177 : GOTO (I124,G2)

E2-> , X T1 G2 .

I178 : GOTO (I125,T1) = I23

I179 : GOTO (I130,X)

D1-> , X .F1
H1-> , X .
F1-> . , Z /.

I180 : GOTO (I130,Z)

D1-> , Z .F1
H1-> . , X / .
F1-> , Z .

I181 : GOTO (I131,D2)

B2-> , Y T1 D2 .

I182 : GOTO (I131, ,)

D2-> , .X T1 F2 / , .Z T1 H2
F2-> , .Z T1
H2-> , .X T1

I183 : GOTO (I131,F2)

C2-> , Y T1 F2 .

I184 : GOTO (I131,H2)

E2-> , Y T1 H2 .

I185 : GOTO (I135,Y)

E1-> , Y .H1
G1-> , Y .
H-> . , X /.

I186 : GOTO (I135,X)

E1-> , X .G1
H1-> , X.
G1-> . , Y / .

I187 : GOTO (I136,E2)

B2-> , Z T1 E2 .

I188 : GOTO (I136,G2)

C2-> , Z T1 G2 .

I189 : GOTO (I36,H2)

D2-> , Z T1 , H2 .
GOTO (I139,+) = I81
GOTO (I139,-) = I82
GOTO (I140,+) = I81
GOTO (I140,-) = I82

I190 : GOTO (I136, ,)

E2-> , .Y T1 H2 / , .X T1 G2
G2-> , .Y T1
H2-> , .X T1

I191 : GOTO (I143, ,)

F-> SIN (E) .
GOTO (I143, *) = I79
GOTO (I143, ,) = I80

I192 : GOTO (I145, ,)

F-> SQR (E) .
GOTO (I145, *) = I79
GOTO (I145, ,) = I80

I193 : GOTO (I146, ,)

F-> COS (E) .
GOTO (I146, *) = I79
GOTO (I146, ,) = I80

I194 : GOTO (I147, ,)

F-> TAN (E) .
GOTO (I147, *) = I79
GOTO (I147, ,) = I80

I195 : GOTO (I148, ,)

F-> ARS (E) .
GOTO (I148, *) = I79
GOTO (I148, ,) = I80

I196 : GOTO (I149, ,)

F-> ARC (E) .
GOTO (I149, *) = I79
GOTO (I149, ,) = I80

I197 : GOTO (I150, ,)

F-> ART (E) .
GOTO (I150, *) = I79
GOTO (I150, ,) = I80

I198 : GOTO (I151, ,)

F-> ABS (E) .
GOTO (I151, *) = I79
GOTO (I151, ,) = I80

I199 : GOTO (I152, ,)

F-> INT (E) .
GOTO (I152, *) = I79
GOTO (I152, ,) = I80
GOTO (I154, B1) = I167
GOTO (I154, ,) = I54
GOTO (I155, B2) = I168
GOTO (I155, ,) = I52
GOTO (I156, C2) = I174

I200 : GOTO (I153, ,)

F-> NEG (E) .
GOTO (I153, *) = I79
GOTO (I153, ,) = I80

I201 : GOTO (I156, ,)

C2-> , .Y T1 F2 / , .Z T1 G2
GOTO (I157, D2) = I181

I202 : GOTO (I157, ,)

C2-> , .X T1 F2 / , .Z T1 H2
GOTO (I158, E2) = I187

I203 : GOTO (I158, ,)

E2-> , .Y T1 H2 / , .X T1 G2

I204 : GOTO (I159,Y)

C1-> , Y .F1

F1-> ., Z / .

I205 : GOTO (I159,Z)

C1-> , Z .G1

G1-> ., Y / .

I206 : GOTO (I160,X)

D1-> , X .F1

F1-> ., Z / .

I207 : GOTO (I160,Z)

D1-> , Z .F1

H1-> ., X / .

I208 : GOTO (I161,Y)

E1-> , Y .H1

H1-> ., X / .

I209 : GOTO (I161,X)

E1-> , X .G1

G1-> ., Y / .

I210 : GOTO (I165,K)

A-> EPP, K , K .

GOTO (I165, ,) = I64

I211 : GOTO (I166,")

A-> DIS, "T2" .

GOTO (I169, C1) = I121

GOTO (I169, T1) = I156

GOTO (I169, ,) = I159

GOTO (I169, DIGIT) = I125

GOTO (I170, D1) = I127

GOTO (I170, T1) = I157

GOTO (I170, ,) = I160

GOTO (I170, DIGIT) = I125

GOTO (I171, E1) = I132

GOTO (I171, T1) = I158

GOTO (I171, ,) = I161

GOTO (I171, DIGIT) = I125

GOTO (I172, F1) = I128

GOTO (I173, G1) = I133

I212 : GOTO (I172, ,)

F1-> , .Z

I213 : GOTO (I173, ,)

G1-> , .Y

I214 : GOTO (I175,Y)

C2-> , Y .T1 F2

G2-> , Y .T1

T1-> .DIGIT T1 / .
GOTO (I179, F1) = I122
GOTO (I179, ,) = I212
GOTO (I180, H1) = I134

I215 : GOTO (I175,Z)

C2-> , Z .T1 G2
F2-> , Z .T1
T1-> .DIGIT T1 / .

I216 : GOTO (I180, ,)

H1-> , .X

I217 : GOTO (I182,X)

D2-> , X .T1 F2
H2-> , X .T1
T1-> .DIGIT T1 / .
GOTO (I185, H1) = I122
GOTO (I185, ,) = I216
GOTO (I186, G1) = I126
GOTO (I183, ,) = I213

I218 : GOTO (I182,Z)

D2-> , Z .T1 H2
F2-> , Z .T1
T1-> .DIGIT T1 / .

I219 : GOTO (I190,Y)

E2-> , Y .T1 H2
G2-> , Y .T1
T1-> .DIGIT T1 / .

I220 : GOTO (I190,X)

E2-> , X .T1 G2
H2-> , X .T1
T1-> .DIGIT T1 / .

I221 : GOTO (I201,Y)

C2-> , Y .T1 F2
T1-> .DIGIT T1 / .

I223 : GOTO (I202,X)

D2-> , X .T1 F2
T1-> .DIGIT T1 / .

I224 : GOTO (I202,X)

D2-> , Z .T1 H2
T1-> .DIGIT T1 / .

I225 : GOTO (I203,Y)

E2-> , Y .T1 H2
T1-> .DIGIT T1 / .

GOTO (I204, F1) = I128
GOTO (I204, ,) = I128
GOTO (I205, G1) = I128
GOTO (I205, ,) = I128
GOTO (I206, F1) = I128
GOTO (I206, ,) = I128
GOTO (I207, H1) = I128
GOTO (I207, ,) = I128
GOTO (I208, H1) = I128
GOTO (I208, ,) = I128
GOTO (I209, H1) = I128
GOTO (I209, ,) = I128

I226 : GOTO (I203,X)

E2-> , X .T1 G2
T1-> .DIGIT T1 / .

I227 : GOTO (I212,Z)

F1-> , Z .

I228 : GOTO (I212,Y)

G1-> , Y .

I229 : GOTO (I214,T1)

C2-> , Y T1 .F2
G2-> , Y T1 .
F2-> ., Z T1 / .
GOTO (I214, DIGIT) = I125

I230 : GOTO (I215,T1)

C2-> , Z T1 .G2
F2-> , Z T1 .
G2-> ., Y T1 / .
GOTO (I215, DIGIT) = I125

I231 : GOTO (I216,X)

H1-> , X .

I232 : GOTO (I217,T1)

D2-> , X T1 .F2
H2-> , X T1 .
F2-> ., Z T1 / .
GOTO (I217, DIGIT) = I125

I233 : GOTO (I218,T1)

D2-> , Z T1 .H2
F2-> , Z T1 .
H2-> ., X T1 / .
GOTO (I218, DIGIT) = I125

I234 : GOTO (I219,T1)

E2-> , Y T1 .H2
G2-> , Y T1 .
H2-> ., X T1 / .
GOTO (I219, DIGIT) = I125

I235 : GOTO (I220,T1)
E2-> , X T1 .G2
H2-> , X T1 .
G2-> ., Y T1 / .
GOTO (I220, DIGIT) = I125

I236 : GOTO (I221,T1)
C2-> , Y T1 .F2
H2-> , Z T1 .
GOTO (I221 DIGIT) = I125

I237 : GOTO (I222,T1)
C2-> , Z T1 .G2
G2-> , , Y T1 / .
GOTO (I222, DIGIT) = I125

I238 : GOTO (I223,T1)
D2-> , X T1 .F2
F2-> . , Z T1 .
GOTO (I223 DIGIT) = I125

I239 : GOTO (I224,T1)
D2-> . , Z T1 .H2
H2-> , X T1 .
GOTO (I224, DIGIT) = I125

I240 : GOTO (I225,T1)
E2-> . , Y T1 .H2
H2-> , X T1 .
GOTO (I225, DIGIT) = I125

I241 : GOTO (I226,T1)
E2-> . , X T1 .G2
G2-> . , Y T1 .
GOTO (I226, DIGIT) = I125
GOTO (I230, G2) = I188
GOTO (I229, F2) = I183

I242 : GOTO (I229, ,)
F2-> , .Z T1

I243 : GOTO (I230, ,)
G2-> , .Y T1
GOTO (I232, F2) = I176
GOTO (I232, ,) = I242
GOTO (I233, H2) = I189

I244 : GOTO (I233, ,)

H2-> , .X T1
GOTO (I234, H2) = I184
GOTO (I234, ,) = I244
GOTO (I235, G2) = I177
GOTO (I235, ,) = I243
GOTO (I236, F2) = I183
GOTO (I236, ,) = I242
GOTO (I237, G2) = I188
GOTO (I237, ,) = I243
GOTO (I238, F2) = I176
GOTO (I238, ,) = I242
GOTO (I239, H2) = I189
GOTO (I239, ,) = I244
GOTO (I240, H2) = I184
GOTO (I240, ,) = I244
GOTO (I241, G2) = I177
GOTO (I241, ,) = I243

I245 : GOTO (I242,Z)

F2-> , Z .T1
T1-> .DIGIT T1 / .

I246 : GOTO (I243,Y)

G2-> , Y .T1
T1-> .DIGIT T1 / .

I247 : GOTO (I244,X)

H2-> , X .T1
T1-> .DIGIT T1 / .

I248 : GOTO (I245,T1)

F2-> , Z T1 .
GOTO (I245, DIGIT) = I125

I249 : GOTO (I246,T1)

G2-> , Y T1 .
GOTO (I246, DIGIT) = I125

I250 : GOTO (I247,T1)

H2-> , X T1 .
GOTO (I247, DIGIT) = I125

LIST OF FOLLOW (A) FOR ALL NON-TERMINALS "A" OF THE GRAMMAR

FOLLOW (S) = {$}
FOLLOW (A) = {)}

FOLLOW (E) = {$,*,/,),+,-}
FOLLOW (A1) = {)}
FOLLOW (A2) = {)}
FOLLOW (B1) = {)}
FOLLOW (A3) = {)}
FOLLOW (A4) = {)}
FOLLOW (B2) = {)}
FOLLOW (C1) = {)} = FOLLOW (D1) = FOLLOW (E1)
FOLLOW (F1) = {)} = FOLLOW (G1) = FOLLOW (H1)
FOLLOW (C2) = {(} = FOLLOW (D2) = FOLLOW (H2)
FOLLOW (I) = {)}
FOLLOW (K) = {), ,}
FOLLOW (T1) = {)}
FOLLOW (T2) = {), , , "}
FOLLOW (R) = {), , , $,*,/,+,- }
FOLLOW (T) = {), $,*,/,+,- }
FOLLOW (E) = {), $,*,/,+,- }

ALGORITHM FOR CONSTRUCTING THE PARSING TABLE :

INPUT :- The canonical collection of sets of items for the augmented grammar G'.

OUTPUT :- The LR parsing table consisting of a parsing action function ACTION and a goto function GOTO.

METHOD :- Let C = { I0 I1 ..... In }. The states of the parser are 0,1...n, state i being constructed from Ii. The parsing actions for state i are determined as follows:

1. If [A->α.aβ] is in Ii and GOTO(Ii,a) = Ij, then set ACTION[I,a] to "shift j". Here a is a terminal.
2. If [A->α.] is in Ii,then set ACTION[I,a] to "reduce A->α" for all 'a' in FOLLOW(A).
3. If [S'->S.] is in Ii , then set ACTION[i,$] to "accept".

   The goto transition for state i are constructed using the rule:
4. If GOTO [Ii,A] = Ij , then GOTO [i,a]=j.
5. All entries not defined by rules (1) through (4) are made "error".
6. The initial state of the parser is the one constructed from the set of items containing [S'-> .S].

The following sheets depict the canonical collection of items and the parsing table.

Data structures used to support the operation of the parser are :

1. The parsing table which consists of five separate files. The appropriate parsing table is loaded at run time.
2. The set of production is also stored in a file and loaded into an array before the program is executed.
3. The stack which is used to guide the parsing action is implemented using a doubly linked list with a special node to indicate the bottom of the stack.

3.DESIGN OF THE CODE GENERATOR :

The code generator (or) expression evaluator is used to evaluate the expression as and when such a need arises.

The evaluator scans the intermediate code(i.e the token list) , and verifies whether expression evaluation is necessary. This case arises when:

1. The evaluator encounters an assignment symbol. It then evaluates the expression to the right of the assignment symbol and updates the corresponding variable which is in the symbol table.
2. A G-statement (for specifying tool movement) uses an expression to specify the value of the co-ordinates. Here, the expression is evaluated and the co-ordinate value calculated.

The expression evaluator generates the output code as G-type statements, omitting line numbers. No variables or loop statements or subroutine calls occur. This format is called ISO G-code.

This standard G-code form can be simulated using commercial simulation software.

HOW ARE THE EXPRESSIONS EVALUATED ?

The expressions occur in the infix form, with unary and binary operators.

The unary operators are : unary "-",NEG, SIN ,COS ,SQR ,TAN ,ARS ,ARC ,ART, ABS and INT.

The binary operators are +,-,/,* . The expression evaluator uses three stacks : a variable stack ,an operator stack and a repeat stack.All these are implemented using doubly linked lists.

The precedence of the operators is as below.

	OPERATOR	PRECEDENCE
1.	Parentheses	1
2.	unary -,NEG	2
3.	SIN ,COS ,TAN ARS ,ARC ,ART ABS ,SQR ,INT	3
4.	* ,/	4
5.	+,-	5
6.	X,Y,Z	6

The associativity for all operators is from left to right. Whenever an operator is encountered ,it is pushed inside the operator stack ,after checking the precedence of the operators already in the stack. If there is any operator of higher priority in the stack ,it is popped out, and the evaluated value is pushed into the variable stack before pushing in the current operator.

Variables and constants encountered are pushed into the variable stack. Whenever an operator is popped ,the required number of variables (unary operator: one and binary operator :two variables) are popped ,the operation carried out and the result pushed back.

A open parentheses is pushed inside the operand stack like other operands. Upon encountering the corresponding close parentheses ,operators are popped and respective operations carried out till the open parentheses operator.

A $ symbol, when encountered, makes the evaluator pop and calculate for all the operators in the stack.

Repeat statements are handled by the following technique:

Upon encountering a repeat statement ,a count is set to the index value. The start header of the loop is pushed into the repeat stack , along with the count.

An end-of-repeat statement causes the top of stack of the repeat stack to be popped ,and the count is decremented by one . This operation is preformed until count equal to zero. Since a stack is used , nested looping is accounted for.

OTHER FUNCTIONS OF THE EVALUATOR :

The evaluator also converts the code written in incremental mode(UIO), into absolute mode(UAO) ,by performing cumulative addition.

The temporary origin mode(UOT) is also converted to the absolute mode (UAO) by adding the difference between the absolute origin(0,0,0) and the temporary origin to the co-ordinate values.

MIR mode is handled by negation of co-ordinate values.

CONCLUSION

This project was completed to the satisfaction of M/S STEER ENGG(P)LTD, and tested and implemented in their factory at Bangalore.

Future expansion of this system is possible in the area of simulation, which is currently done using commercial simulation software.

1. INTRODUCTION

SYSTEM OVERVIEW

8600 MC is a numerical control that can be applied to machining centres, turning centers, special machines and FMS's.

HARDWARE CONFIGURATION

The control is made up of two basic components:

- a control panel with 9" CRT
- a 10 or 20-slot controller rack.

CONTROL PANEL

The control panel is the man-machine interface and allows the operator to enter commands, compile programs and monitor the machine performance.

CONTROLLER RACK

The rack accommodates the system CPU and the modules for managing axes and digital and analog I/O. It is connected to the control panel by means of a serial line.

SOFTWARE CONFIGURATION

The software of the 8600 MC can be broken down into the following subsystems

- system software
- process software
- interface software

SYSTEM SOFTWARE

The system software has been developed by our collaborators M/s. OSAI A-E of Italy. It is resident in EPROM and controls the machine functions, the dialogue with the operator and with the system peripherals.

PROCESS SOFTWARE.

This subsystem allows the control to perform the following functions:

- data decodification
- axes interpolation
- position control
- management of machine functions
- management of I/O functions.

The machining program can be input from keyboard or loaded from a peripheral (tape reader, cassette or personal computer). After the program has been loaded into the system memory, you can launch it with a specific command.

INTERFACE SOFTWARE (SIPROM)

SIPROM can be used by machine tool manufactures for developing the software interface between the control and the machine tool.

The software interface can then be stored on the system CMOS RAM or on EPROM.

2. OPERATOR PANEL

OPERATOR PANEL

This version features:

- an alphanumeric keyboard
- a functional keyboard
- a series of pushbuttons, selectors and switches
- a 9" CRT

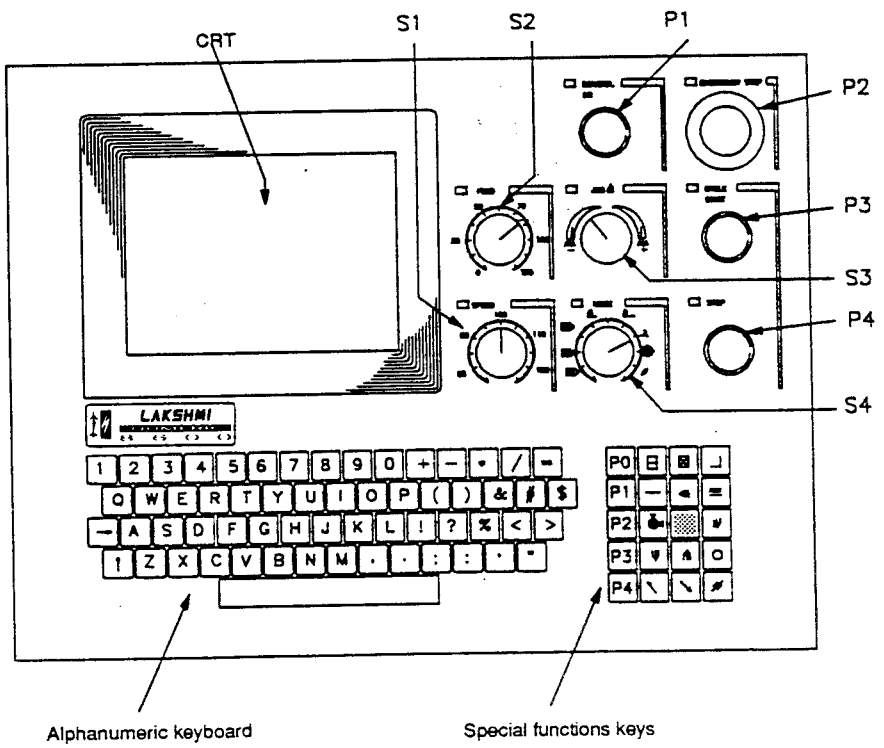


Figure 2.1 Operator Panel

6. PROGRAMMING

This chapter shows you the programming functions available with your contro..

AXIS MOTION

The control governs the axes according to the specifications of EIA RS-267. This standard defines the motion of the tool in relation to the workpiece, independent of which - axes or workpiece - is actually moving, as shown in figure 6.1.

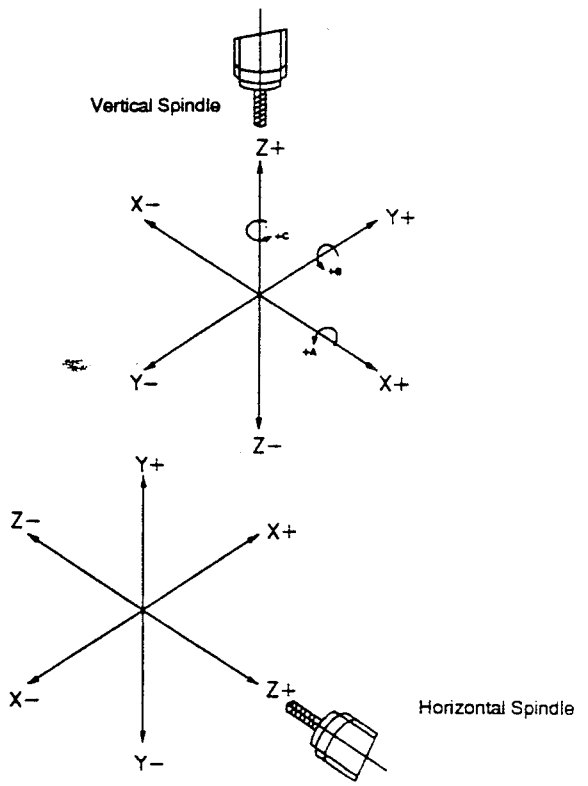


Figure 6.1 Vertical and Horizontal Spindle

PROGRAM BLOCKS WITH G FUNCTIONS

Preparatory G functions range from G00 to G99. At present, only the codes shown in Table 6.2 are available with the control.

Within a program block, always insert the G function immediately after the block number (if present) and before any other operand.

You can program a G operand either explicitly or implicitly, by means of an E parameter (with a variable of the byte type).

We have classified G functions in thirteen model groups. They are listed in Table 6.4.

A block can include several G functions, provided they are mutually compatible. Compatibility between the different groups is dealt with in Table 6.5.

Table 6.4 - Modal Groups

Group	G Functions	Description
a	G00-G01-G02-G03-G33	Definition of the movement
b	G17-G18-G19	Definition of interpolation plane
c	G27-G28-G29	Definition of the dynamic mode (point-to-point or continuous)
d	G21-G20	Open and close GTL programming ambient
e	G40-G41-G42	Tool radius offset enable and disable
f	G70-G71	Programming in alternative unit mm/inch
g	G81-G86-G89-G80	Standard canned cycles
h	G90-G91	Incremental/absolute mode
i	G79	Programming referred to machine zero
j	G04-G09	Attributes of the dynamic mode
k	G72-G73-G74	Measuring cycles
l	G93-G94-G95	V/D feedratecycles
m	G96-G97	Spindle speed

DEFINITION OF THE TYPE OF MOVEMENT

The allowable functions are:

G00 rapid axes positioning

G01 linear interpolation

G02 circular interpolation CW

G03 circular interpolation CCW

G33 constant or variable pitch threading.

Key to symbols:

[] enclose optional elements:

{} enclose alternative elements.

Intermediate zeros can be omitted.

for example, G00 = G G01 = G1

PARAMETRIC PROGRAMMING

You can use E parameters for the geometrical and technological data of a machining cycle, E parameters allow mathematical and trigonometric operations and calculations of expressions as well.

The maximum number of E parameters must be defined during system configuration.

E parameters require different indexes for variables having different format the allowable formats are shown in table 6.8

Table 6.8 - E Parameters and their Formats

Format	Parameters	Min/max value
BY (byte)	E0..E9	0 to 255
IN (integer)_	E10..E19	-32768 to +32767
LI (long integer)	E20..E24	-2.147.483.647 to +2.147.483.647
RE (real)	E25..E29	±7 whole or decimal digits
LR (longreal)	E30..(*)	±16 significant whole and decimal digits ±13 whole digits

(*) Maximum number of E parameters defined during configuration

E parameters receive values in special assignment block is:

EN= <expression>

Where

<expression> Can be a numerical value or a mathematical expression whose result will be stored in the E parameter having index n.

n can be a number or an E parameter (of either byte or integer type).

For example:

E(E5) where E5 = n

EXPRESSIONS

An <expression> is a mathematical equation formed by arithmetic operators, functions and operands (E parameters or numerical constants). Arithmetic operators are:

+ (addition)
- (Subtraction)
* (multiplication)
/ (division).

Possible functions are:

SIN(A)	sine of A
COS(A)	cosine of A
TAN(A)	tangent of A
ARS(A)	arcsine of A
ARC(A)	arccosine of A
ART(A)	arctagent of A
SQR(A)	square root of A
ABS(A)	absolute value of A
INT(A)	integer of A
NEG(A)	inverts the sign A

PROGRAM BLOCKS WITH THREE-LETTER COMMANDS

This section describes the function and syntax of program blocks that use three-letter commands.

We have grouped three-letter commands in seven classes. You can use them for:

- modifying the reference system of the axes
- modifying the sequence of program execution
- performing miscellaneous commands
- performing I/O commands
- monitoring tool life
- managing a probe
- managing tool offsets.

Modifying the Reference System of the Axes

The commands in this class allow you to change the cartesian reference system in which you programmed a profile.

The following commands belong to this class:

UAO - use absolute origins

UOT - use temporary origins

UIO - use incremental origins

MIR - mirror machining

URT - rotation of the plane

SCF - scale factor

RQO - requalify origins

USE ABSOLUTE ORIGINS - UAO

This command allows you to activate the absolute origins previously defined with AXO.

The allowable format is:

(UAO,n[,VAR-1,VAR-2...VAR-n])

Where:

n defines the number of the permanent origin to be selected. It can be either a numerical constant or an E parameter of the integer type (E10 to E19).

VAR -I is a character that represents the name of the axis for which the origin n must be specified.

For undeclared axes, the current origin stays in effect.

If no axis name is specified, the origin n is activated for all those axes in which the n origin has been declared.

Example:

(UAO,1)	- activate absolute origin 1
.....	- program referred to origin 1 for all axes
(UAO,2,X,Y)	- activate absolute origin 2 for axes X and Y
(UAO,3,Z)	- activate origin 3 for axis Z
.....	- program referred to origin 2 for axes X and Y to origin 3 for axis Z to origin 1 for all the other axes
(UAO,0)	- reactivate absolute origin zero for all axes

Notes

a. At power-up and after a reset, the control automatically establishes absolute origin zero for all the axes.

b. You can specify as many as 8 axis names. An axis can only be declared once in each UAO command.

c. To declare a different origin for each axis, program as many UAO blocks as the desired origins.

d. If the selected origin had been stored in the file in the alternative measuring unit, i.e. preceded by a "-" sign, the system automatically converts its value dividing or multiplying it by the conversion factor.

USE TEMPORARY ORIGINS -UOT

Temporarily shifts the absolute origin by a programmed distance. The allowable format is:

(UOT,0,VAR-1[,VAR-2])

Where:

n defines the number of the absolute origin you want to temporarily modify

VAR-I is an axis and a dimension.
The control takes the dimension as an absolute offset and adds it to the value of the absolute origin for that axis.
For undeclared axes, the current origin stays in effect.

Example (Fig.6.97):

(UAO,0)	- activates absolute origin 0
.....	- program referred to absolute origin 0 for all axes
.....	
(UOT,0,X100,Y100)	- applies a temporary origin to origin 0, with X100 and Y 100 offset (Point 1)
.....	- this portion of the program uses the temporary origin
.....	
(UOT,1,X-250,Y50)	- applies a temporary origin to origin 1, with X-250 and Y50 offset (point 2)
.....	- this portion of the program uses X-250 Y50 temporary origin
.....	
(UAO,0)	- reactivates absolute origin 0 for all axes

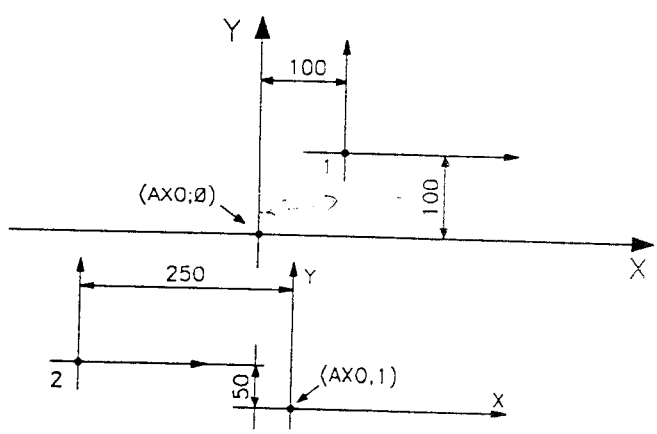


Figure 6.97 Using UOT command

Notes

a. With the UOT command you must declare atleast one axis and as many as 8 axes. An axis can only be declared once in each UOT command.

b. A temporary origin remains in effect until you redefine it with a new UCT or you reestablish the absolute origin with either UOA or RESET.

c. You must program the dimension in the UOT command with the current measuring unit (G70/G71). If set, the control will apply the scale factor to the temporary origin.

USE INCREMENTAL ORIGINS - UIO

This command allows an incremental shift of the current origin for each axis specified in the command.

The allowable format is:

(UIO,VAR-1[,VAR-2...VAR-n])

Where:

VAR-I is an axis and a dimension
The control takes the dimension as an absolute offset and adds it to the value of the absolute origin for that axis.
For undeclared axes, the current origin stays in effect.

Example (Fig.6.98):

```
.....
N65   (UIO,X20,Y20)    - point 1
.....
N121  (UIO,Y-40)       - point 2
.....
N180  (UIO,X-45)       - point 3
.....
N230  (UIO,Y35)        - point 4
.....
N300  (UAO,0)
```

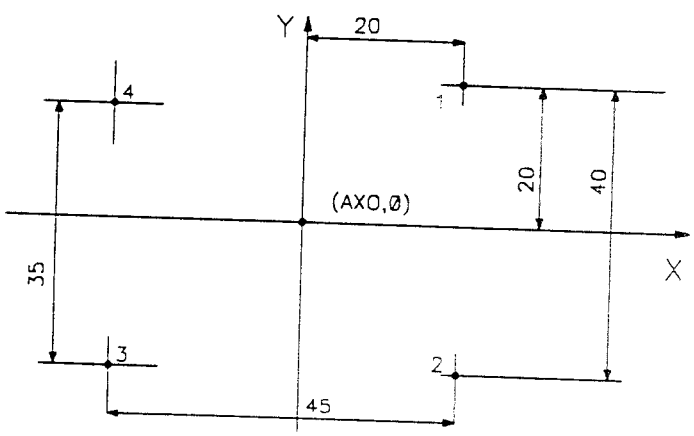


Figure 6.98 Using UIO command

Notes

a. With the UIO command you must declare atleast one axis and as many as 8 axes. An axis can only be declared once in each UIO command.

b. An incremental origin remains in effect until you redefine it with a new UIO or restablish the absolute origin with either UOA or RESET.

c. You must program the dimension in the UIO command with the current measuring unit (G70/G71). If set, the control will apply the scale factor to the temporary origin.

MIRROR MACHINING - MIR

The MIR command reverse (mirrors) the programmed direction of motion for the axes specified in the command.

The allowable format is:

(MIR[,VAR-1,...,VAR-n])

where:

VAR-n is a letter that corresponds to a configured axis in the system

Example (Fig.6.98 has been suppressed):

```
N24 (MIR,Z)
.....
N42 (MIR,Z,X)
.....
N84 (MIR,X)
.....
N99 (MIR)
```

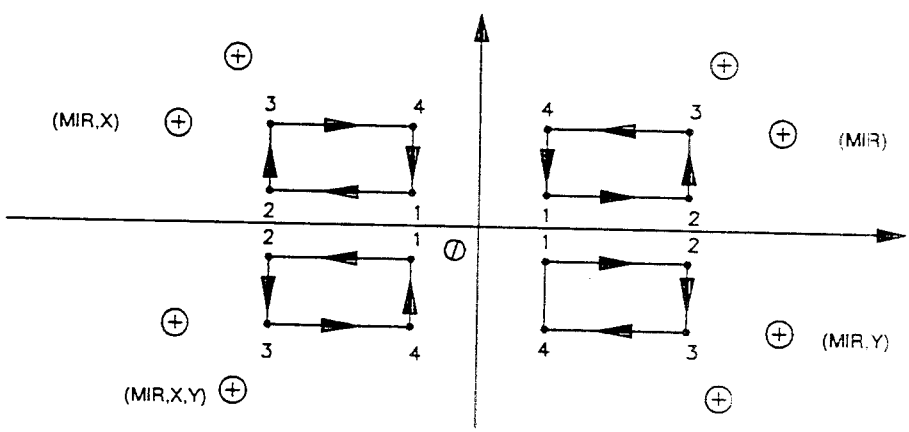


Figure 6.99 Use of the MIR command

Notes

a. The control mirrors programmed axis move with respect to the current origin.

b. For the undeclared axes the preceding MIR command remains in effect.

c. You can declare as many as 8 axes. If no axes are programmed in the MIR command, the mirror function is deactivated for all configured axes.

d. The control applies the mirror function to an axis beginning with the first movement of that axis after the MIR command.

e. Rotation and mirror commands (respectively, URT and MIR) are applied in the following order: URT first, MIR second.

REPEAT A SET OF PROGRAM BLOCKS - RPT/ERP

RPT and ERP delimit a set of program blocks that should be executed a specified number of times. The set begins with RPT and ends with ERP.

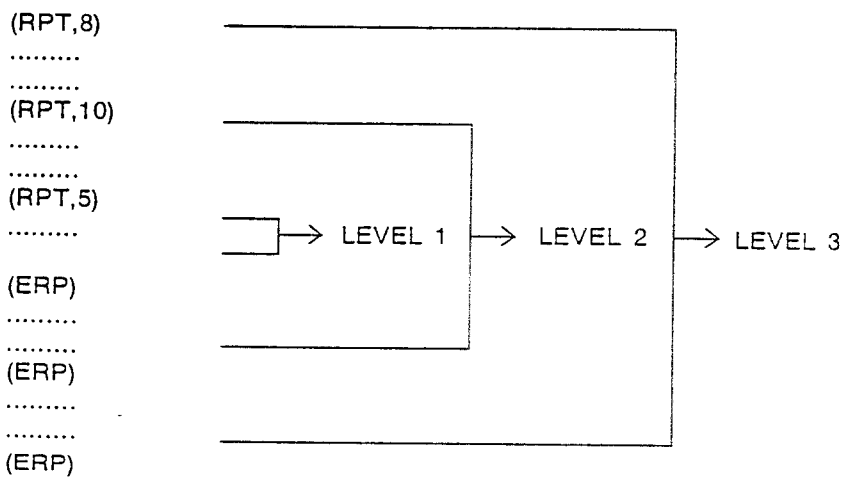
The format of the full commands is:

(RPT,n) - repetition command and number of times.
.....
..... - set of blocks to be executed the specified number of times
.....
.....
(ERP) - defines the end of the set.

where:

n is the number of executions. n must be a whole number from 1 to 99. You can program n explicitly or implicitly, with a byte type E parameter (E0 to E9).

The control allows 3 repeat levels. You can program up to 2 repeat commands inside another repeat command (Figure 6.95).



Figures 6.105 Repeat levels

The control allows 3 repeat levels. You can program up to 2 repeat commands inside another repeat command.

USING SUBROUTINES - CLS

The CLS command allows you to call and execute a subroutine that is stored in memory. A subroutines is a sequence of blocks that define a machining cycle. The allowable format is:

(CLS,FILE NAME[/DEVICE])

where:

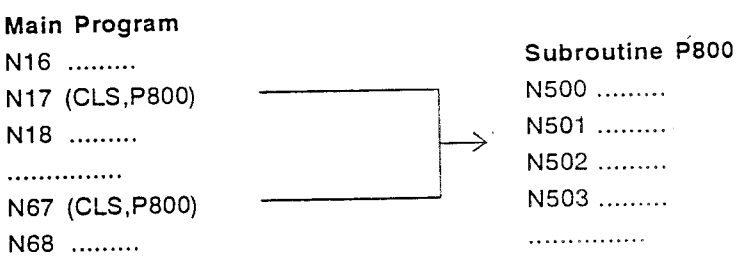
FILE NAME is the name of the subroutine file to be recalled. It can have as many as 6 alphanumeric characters. The first character must be a letter. All letters must be capital letters.

/DEVICE is the name of the device containing the program. Use/to separate the device name from the file name. The device name can have 2 or 3 alphanumeric characters. The first character must be a letter. All letters must be capital letters. If the device is not specified, the control defaults the device declared during configuration.

For example:

N1 (CLS,P800/P2) calls and executes subroutine P800 allocated on memory MP2. If MP2 is the default memory,it does not need to be programmed in the CLS command.

Example of a Subroutine Call:

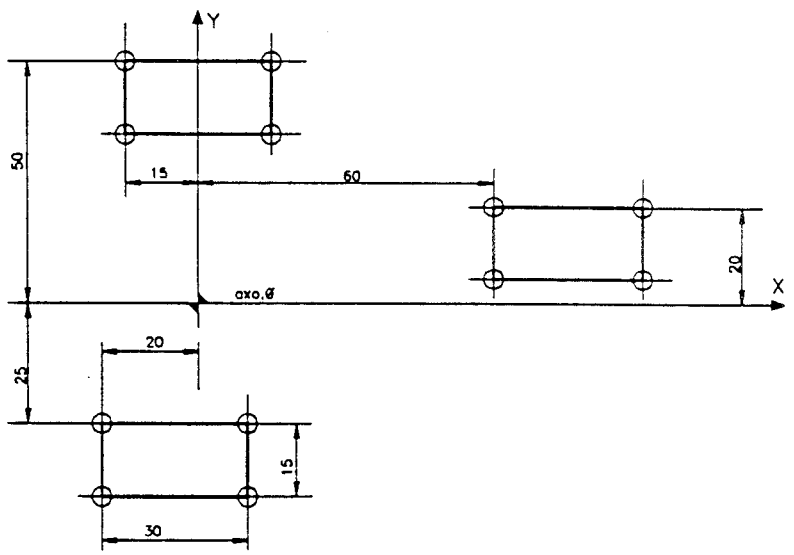


Notes

a. Only two call levels are available, i.e., the program called by CLS can call up other programs but the called programs cannot call up other programs.

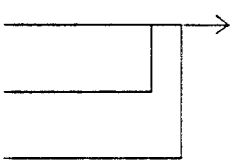
b. Subroutines can be parametric, the numeric values of the parameters are defined in the main program during the recall.

Figure 6.106 shows another examples for subroutine programming.



Main program

```
N19 (DIS,"...")
N20 S2000 F180 T.2.2 M6
N21 (UOT,0,X-20,Y-25)
N22 (CLS,S600)
N23 (UOT,0,X-15,Y50)
N24 (CLS,S600)
N25 (UOT,0,X60,Y20)
N26 (CLS,S600)
N27 Z...
```



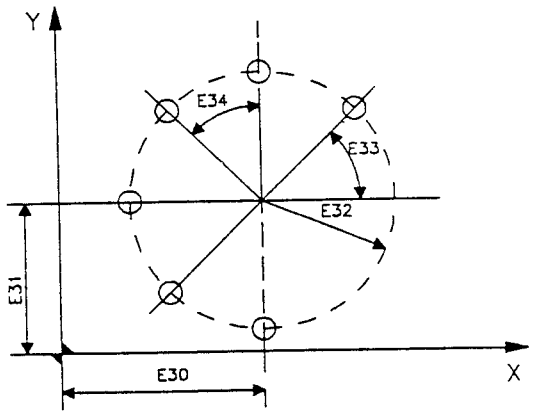
Subroutine S600

```
N501  G81 R... Z...
N502  XY
N503  Y-15
N504  X30
N505  Y
N506  G80
```

Figure 6.106 Drilling Jig (30 x 15)

PARAMETRIC SUBROUTINES

In a parametric subroutine, the numeric values of the parameters (G, S, X, Z, F, etc.) are defined in the main program during the recall. Figure 6.107 shows an example of parametric subroutines.



E1 = No. of holes
E2 = fixed cycle G
E30 = X of the grid centre
E31 = Y of the grid centre
E32 = grid radius
E33 = starting angle
E34 = angular pitch
E36 = R dimension
E37 = Z dimension

```
.............
N43 (DIS,"TWIST DRILL D12")
N44 T2.2 M6 S900 F70 M3
N45 E30=50
N46 E31=50
N47 E32=30
N48 E33=45
N49 E34=45
N50 E1=6
N51 E36=2
N52 E37=-10
N53 E2=81
N54 (CLS,SUB800)
N55 G Z20 M5
.............
```

Subroutine SUB800

```
N799 (DIS,"CIRCULAR GRIDS")
N800 (UOT,0,XE30,YE31)
N801 E25=E33
N802 (RPT,E1)
N803 (URT,E25)
N804 GE2 RE36 ZE37
N805 XE32 Y
N806 E25=E25+E34
N807 G80
N808 (ERP)
N809 (UAO,0)
N810 (URT,0)
```

Figure 6.107 Circular grid

For further information, refer to the "Parametric programming" section.

```
# include <stdio.h>
# include <alloc.h>
# include <string.h>
# include <stdlib.h>
# include "symbol.h"
# include  "evalu.c"
    struct tlist head[50],*node,*temp;
    struct symb *hsym, *sym,*sear;
    int j,rcount,scount,cflag=0;
    char inp[100];
 int chcount =0;
 int fileno=0;
 FILE *fptr1;
 struct stk
  {
    struct stk *prev;
    int item;
    struct stk *next;
  };
 struct stac
  {
    struct stac *prev;
    FILE *item;
    struct stac *next;
   };


 struct stk *start,*s_top ,*tempr;
 struct stac *shead,*snode,*stemp;
 struct type_ele table[51][68];
 struct  type_prod prod[98];
 FILE *fptr3,*fptr4;

 /* Routine to load appropriate parsing table */

 void  filload(void)
 {
    int row=0,col=0,cou=49;
    if(fileno == 5)
     cou=50;
    while (row <=cou)
    {
    while (col <= 67)
      {
      fread(&table[row][col],sizeof(struct type_ele),1,fptr3);
      ++col;
            }
      col=0;
```

```
    ++row;
  }
    fclose(fptr3);
}

/* Routine to load details of the productions */

 void  proload(void)
 {
   int co=0;
   fptr4=fopen("prodc.dat","r");
   while(co < 98)
     {
       fread(&prod[co],sizeof(struct type_prod),1,fptr4);
       ++co;
      }
 fclose(fptr4);
 }

 void error()
 {
  printf("\n SYNTAX ERROR IN LINE :%d \n",j+1);
  cflag=1;
  return;
 }

 /* Routine to push tokens into the LR parser's stack */

 void push(int n)
 {
   s_top->next=malloc(sizeof(struct stk));
   s_top->next->prev = s_top;
   s_top = s_top->next;
   s_top->next = NULL;
   s_top->item = n;
  }

 /* Routine to pop tokens from the LR parser's stack */

  void pop(void)
  {
     if ( s_top == start )
       {
         printf(" \n STACK EMPTY ! \n");
         cflag=1;
          while(node != NULL)
           node=node->next;
         return;
```

```
    }
   else
    {
     tempr=s_top->prev;
     s_top->prev->next=NULL;
     s_top->prev=NULL;
     free(s_top);
     s_top=tempr;
    }
}

/* Routine to store file pointer on encountering subroutine calls */
void spush(void)
{
 snode->next=malloc(sizeof(struct stac));
 snode->next->prev = snode;
 snode = snode->next;
 snode->next = NULL;
 snode->item = fptr1;
}

/* Routine to retrieve the file pointer after processing subroutine */

 void spop(void)
 {
      stemp=snode->prev;
      snode->prev->next=NULL;
      snode->prev=NULL;
      free(snode);
      snode=stemp;
  }

/* Stack initialization is performed here */

void initial(void)
{
 start=malloc(sizeof(struct stk));
 start->prev=NULL;
 start->item=0;
 start->next=NULL;
 s_top = start;
 }

/* Routine to check whether the appropriate parsing table is loaded */

 void filchk(void)
    {
      int x;
```

```
x = s_top->item / 50 +1;
if (fileno != x)
    {
     switch(x)
       {

         case 1 :
                  fptr3=fopen("table1.dat","rb");
                  filload();
                  fileno = 1;
                  break;

         case 2 :
                  fptr3=fopen("table2.dat","rb");
                  filload();
                  fileno = 2;
                  break;

         case 3 :
                  fptr3=fopen("table3.dat","rb");
                  filload();
                  fileno = 3;
                  break;

         case 4 :
                  fptr3=fopen("table4.dat","rb");
                  filload();
                  fileno = 4;
                  break;

         case 5 :
                  fptr3=fopen("table5.dat","rb");
                  filload();
                  fileno = 5;
                  break;

         default :
                  exit(0);
                  break;
       } /* switch */
    } /* if */

}

/* LR PARSER */

void  parser(void)
{
```

```
 int row,col,t,splpush;
/* t is for calculating the no. of times you have to pop from the stack */
 proload();
 initial();
 node=head[j].next;
 while(node != NULL)
  {
    if(node->flag)
    {
    filchk();

    row = s_top->item - ( (fileno-1) * 50);
    col=node->typ-1;
    switch(table[row][col].c)
 {
 case 's':
 case 'S':
        {
         push(node->typ);
         push(table[row][col].i);
         node=node->next;
         /*tempr = start;
         while(tempr != NULL)
         {
           printf("\t%d  ",tempr->item);
           tempr=tempr->next;
          }
         printf("\n\n");*/
         break;
        }
  case 'r':
  case 'R':
          t= prod[ (table[row][col].i) -1 ].beta * 2;
          while(t>0)
          {
           pop();
           --t;
          }

          splpush=prod[ (table[row][col].i) -1].c;
          filchk();
          push(splpush);
          row = s_top->prev->item - (fileno -1)*50;
          col = s_top->item - 1;
          if(table[row][col].c =='g'|| table[row][col].c == 'G')
              push(table[row][col].i);
          else
            error();
```

```
        /*tempr = start;
        while(tempr != NULL)
         {
           printf("\t%d  ",tempr->item);
           tempr=tempr->next;
         }
         printf("\n\n");*/
         break;
case 'e':
case 'E':
        /*  give error message depending on integer value in structure */
        error();
        return;
case 'a':
case 'A':
         node=node->next;
         pop();
         pop();
         initial();
         break;
default :
         printf("\nHello,have a nice day !");
         printf("\nBut first tell me what's happening you  @#@# \n");
         cflag=1;
         return;
}
}
else
 node=node->next;
}
return;

}

/* Routine to search whether the E-variable encountered is already
   present in the symbol table */

 int search(int num)
 {
    sear = hsym;

    while (sear != NULL)
        {
         if(sear->evar == num)
           return(1);
          sear=sear->next;
      }
     return 0;
```

```
  }
/* New node is created here,for the token encountered */

 void create(int n)
 {
       node->next = malloc(sizeof(struct tlist));
       node->next->prev = node;
       node = node->next;
       node->typ=n;
       node->next=NULL;
       return;
   }
main()
  {
     int flag=0;
     int gflag=0;
     int first_after_g =0;
     int err_flag=0;
     char * token;
     int i,count,a;
     char c,infile[11],outfile[11];
     char ch[6];
     fflush(stdin);
     printf("\n\tEnter the input filename  :");
     gets(infile);

   /* LEXICAL ANALYSER */

     if( (fptr1=fopen(infile,"r")) == NULL)
         {
          printf("\nUnable to open %s",infile);
          exit(0);
          }
     hsym=malloc(sizeof(struct symb));

     hsym->next=NULL;
     sym=hsym;
     shead=malloc(sizeof(struct stac));
     shead->prev=NULL;
     shead->next=NULL;
     shead->item=0;
     snode=shead;
     j=0;

     while(1)
     {
      flag=0;
```

```
while (!feof(fptr1))
    {
     err_flag =0;
     i=0;
     do
     {
      inp[i]=getc(fptr1);
       ++i;
     }
     while(  (inp[i-1] != '\n') && (!feof(fptr1)) );
  if(inp[i-1] == '\n')
     inp[i-1]='\0';
  if(feof(fptr1))
  {
     inp[0]='\0';
      flag=1;
   }
  count=i;
  i=0;
  head[j].prev = NULL;
  head[j].next = NULL;
  node = &head[j];

while (count > 1)
   {
 if(!err_flag)
 {
 switch(inp[i])
  {

  case ' ':
           while( (inp[i] == ' ') && (count != 0) )  /* skip blank spaces */
             {
                   --count;
                   ++i;
              }
            break;

  case '0':
  case '1':
  case '2':
  case '3':
  case '4':
  case '5':
  case '6':
  case '7':
  case '8':
  case '9':
```

```
  while( ( inp[i] <= '9' ) && ( inp[i] >= '0' )  &&
         (count != 0) )
        {
         create(21);
         node->flag=1;
         node->d = inp[i];
         --count;
         ++i;

         }
   break;

case 'e':
case 'E':
     if( ( inp[i+1] <= '9' ) && ( inp[i+1] >= '0' ) )
     {
      if( (inp[i+2] <= '9')  &&  (inp[i+2] >= '0') )
      {
      ch[0]=inp[i+1];
      ch[1]=inp[i+2];
      ch[2]='\0';
      i+=3;
      count-=3;
      }
      else
      {
      ch[0] = '0';
      ch[1]=inp[i+1];
      ch[2]='\0';
      i+=2;
      count-=2;
      }

      create(2);
      node->flag=1;
      if(search(atoi(ch))==0)
      {
          node->stloc=malloc(sizeof(struct symb));
          node->stloc=sym;
          sym->evar = atoi(ch);
          if (sym->evar >= 0 && sym->evar <= 19)
             sym->value.ival =0;
          else
            if (sym->evar <= 24)
               sym->value.lival = 0;
          else
               sym->value.fval = 0;
```

```
        sym->next=malloc(sizeof(struct symb));
        sym=sym->next;
        sym->next=NULL;
       }
       else
       {
        node->stloc=malloc(sizeof(struct symb));
        node->stloc=sear;
       }
      break;
     }
     else
     if ( (( inp[i+1] == 'r' ) || ( inp[i+1] == 'R')) &&
          (( inp[i+2] == 'p' ) || ( inp[i+2] == 'P')) )
        {
          i+=3;
          count-=3;
          --rcount;
          create(15);
          node->flag=1;
          if ( rcount<0)
          {
           printf("\n Missing RPT for ERP in LINE %d",j+1);
           cflag=1;
          }
          break;
        }
      else
      if ( (( inp[i+1] == 'p' ) || ( inp[i+1] == 'P')) &&
           (( inp[i+2] == 'p' ) || ( inp[i+2] == 'P')) )
         {
           i+=3;
           count-=3;
           create(17);
           node->flag=1;
           break;
         }
       else
        {
         printf("LINE %d :Possible misspelt variable / \"erp / epp\" at position %d
",j+1,i+1);
         cflag=1;
         i+=3;
         count-=3;
         err_flag=1;
         break;
         }
```

```
case 'g':
case 'G':
        ++i;
        --count;
        if( inp[i] <= '9' && inp[i] >= '0')
          {
           chcount =0;
           while( (inp[i] <= '9') && (inp[i] >= '0'))
            {
            ch[chcount]=inp[i];
            ++chcount;
            ++i;
            --count;
            }
            ch[chcount]='\0';

         create(atoi(ch));
         node->d='g';
        node->flag=0;
       }
        gflag = 1;
        break;

 case '(':
    ++i;
    --count;
    create(1);
    node->flag=1;
    break;


 case ')':
     ++i;
     --count;
     create(38);
     node->flag=1;
     break;

case '+':
     ++i;
     --count;
     create(36);
     node->flag=1;
     break;

case '-':
        ++i;
        --count;
```

```
        create(37);
        node->flag=1;


      while( ( (inp[i] == ' ') || (inp[i] == '-')) && (count != 0) )

            {
              if (inp[i] == '-')
                 printf("LINE %d extra minus sign in position %d",j+1,i+1);
             cflag=1;
              err_flag=1;
              --count;
              ++i;
             }

    break;

case '*':
    ++i;
    --count;
    create(34);
    node->flag=1;
    break;

  case '/':
        ++i;
        --count;
        create(35);
        node->flag=1;
        break;
case '"':
        ++i;
        --count;
        create(19);
        node->flag=1;
        while((inp[i]!='"') && (count!=0))
        {
         if(inp[i]==' ')
         {
           ++i;
           --count;
         }
         else
         {
           create(22);
           node->flag=1;
           ++i;
           --count;
```

```
            }
          }
          if(inp[i]==''')
          {
          create(19);
          node->flag=1;
          ++i;
          --count;
          }
         break;
case '=':
          ++i;
          --count;
          create(39);
          node->flag=1;
          break;
case '.':
          create(23);
          node->flag=1;
          node->d=inp[i];
          ++i;
          --count;
          break;
case ',':
          ++i;
          --count;
          create(20);
          node->flag=1;
          break;
case 'x':
case 'X':
          ++i;
          --count;
          if (gflag && first_after_g)
          {
            create(40);
            node->flag=1;
            }
          create(3);
          node->flag=1;
          first_after_g =1;
          break;
case 'y':
case 'Y':
          ++i;
          --count;
          if (gflag && first_after_g)
          {
```

```
            create(40);
            node->flag=1;
           }
          create(4);
          node->flag=1;
          first_after_g =1;
          break;
case 'z':
case 'Z':

          ++i;
          --count;
          if (gflag && first_after_g)
          {
            create(40);

           }
          create(5);
          node->flag=1;
          first_after_g =1;
          break;
 case 'S':
 case 's':
          if ( (( inp[i+1] == 'I' ) || ( inp[i+1] == 'i')) &&
               (( inp[i+2] == 'N' ) || ( inp[i+2] == 'n')) )
            {
               i+=3;
               count-=3;
               create(24);
               node->flag=1;
               break;
             }
            else
            {
            if ((( inp[i+1] == 'Q' ) || ( inp[i+1] == 'q')) &&
               (( inp[i+2] == 'R' ) || ( inp[i+2] == 'r')) )
             {
               i+=3;
               count-=3;
               create(25);
               node->flag=1;
               break;
               }
              else
              {
               if((( inp[i+1] == 'c' ) || ( inp[i+1] == 'C')) &&
                 (( inp[i+2] == 'F' ) || ( inp[i+2] == 'f')) )
                 {
```

```
                i+=3;
                count-=3;
                create(12);
                node->flag=1;
                break;
                }
               else
               {
              printf("LINE %d :Syntax error, possibly misspelt \"sin (or) sqr (or) scf\"
at position %d" ,j+1,i+1);
              cflag=1;
              i+=3;
              count-=3;
              err_flag=1;
              break;
             }
            }
            }
  case 'C':
  case 'c':
          if ( (( inp[i+1] == 'O' ) || ( inp[i+1] == 'o')) &&
               (( inp[i+2] == 'S' ) || ( inp[i+2] == 's')) )
           {
              i+=3;
              count-=3;
              create(26);
              node->flag=1;
              break;
           }
          else
          {
            if ( (( inp[i+1] == 'L' ) || ( inp[i+1] == 'l')) &&
               (( inp[i+2] == 'S' ) || ( inp[i+2] == 's')) )
            {
              i+=3;
              count-=3;
              create(16);
              node->flag=1;
              chcount=0;
              while((inp[i]!=')') && (count!=0))
            {
            if(inp[i]==' ')
            {
            ++i;
            --count;
            }
            else
            if(inp[i]==',')
```

```
            {
             create(20);
             node->flag=1;
             ++i;
             --count;
             }
            else
            {
              ch[chcount]=inp[i];
              create(22);
              node->flag=1;
              ++chcount;
              ++i;
              --count;
             }
            }
            ch[chcount]='\0';
            spush();
            if((fptr1=fopen(ch,"r"))==NULL)
            {
             printf("\n Unable to open subroutine file\n");
             exit(0);
            }
            if(scount >= 2)
                {
                 printf("\nMore than two nested subroutines not possible");
                 exit(0);
                }
              else
                ++ scount;

              break;
           }
          else
           {
             printf("LINE %d: Syntax error,possibly misspelt \"cos (or) cls\" at
position %d" ,j+1,i+1);
             cflag=1;
             i+=3;
             count-=3;
             err_flag=1;
             break;
             }
           }
  case 'T':
  case 't':
         if ( (( inp[i+1] == 'A' ) || ( inp[i+1] == 'a')) &&
              (( inp[i+2] == 'N' ) || ( inp[i+2] == 'n')) )
```

```
            {
               i+=3;
               count-=3;
               create(27);
               node->flag=1;
               break;
               }
             else
             {
             printf("LINE %d:Syntax error,possibly misspelt \"tan\" at position %d"
,j+1,i+1);
             cflag=1;
             i+=3;
             count-=3;
             err_flag=1;
             break;
            }

    case 'a':
    case 'A':
            if(( inp[i+1] == 'r' ) || ( inp[i+1] == 'R'))
            {
                if(( inp[i+2] == 's' ) || ( inp[i+2] == 'S'))
                 {
                   i+=3;
                   count-=3;
                   create(28);
                   node->flag=1;
                   break;
                 }
                 else
                 {
                   if(( inp[i+2] == 'c' ) || ( inp[i+2] == 'C'))
                     {
                       i+=3;
                       count-=3;
                       create(29);
                       node->flag=1;
                       break;
                      }
                     else
                      {
                       if(( inp[i+2] == 'T' ) || ( inp[i+2] == 't'))
                        {
                          i+=3;
                          count-=3;
                          create(30);
                          node->flag=1;
```

```
                    break;
                  }
                }
              }
            }
          else
          {
            if( (( inp[i+1] == 'b' ) || ( inp[i+1] == 'B'))&&
                (( inp[i+2] == 's' ) || ( inp[i+2] == 'S')))
              {
               i+=3;
               count-=3;
               create(31);
               node->flag=1;
               break;
              }
              else
              {
               if(inp[i+1]<='9' && inp[i+1]>='0')
               {
                 chcount=0;
                 ++i;
                 --count;
                while(((inp[i]<='9' && inp[i]>='0') || inp[i] == '.')&& count
!=0)
                {
                  ch[chcount]=inp[i];
                  ++chcount;
                  ++i;
                  --count;
                }

                if (node->typ == 20)
                  {

                    temp = node->prev;
                    node->prev->next = NULL;
                    node->prev = NULL;
                    free(node);
                    node = temp;
                    }
                  create(atoi(ch));
                node->flag=0;
                node->d='a';
               break;
               }
               else
               {
```

```
                printf("LINE %d:Syntax error, possibly misspelt \"ars,arc,art,abs\"
at position %d" ,j+1,i+1);
                cflag=1;
                i+=3;
                count-=3;
                err_flag=1;
                break;
                }
             }
           }

  case 'i':
  case 'I':
       if( (( inp[i+1] == 'n' ) || ( inp[i+1] == 'N' )) &&
          (( inp[i+2] == 'T' ) || ( inp[i+2] == 't' )) )
           {
              i+=3;
              count-=3;
              create(33);
              node->flag=1;
              break;
           }

           else
            {
               printf("LINE %d:Syntax error,possibly misspelt \"int\" at position
 %d" ,j+1,i+1);
                cflag=1;
                i+=3;
                count-=3;
                err_flag=1;
                break;
             }
   case 'n':
   case 'N':
           if((( inp[i+1] == 'e' ) || ( inp[i+1] == 'E'))&&
              (( inp[i+2] == 'G' ) || ( inp[i+2] == 'g')))
            {
              i+=3;
              count-=3;
              create(32);
              node->flag=1;
              break;
              }

           else if( inp[i+1] <= '9' && inp[i+1] >= '0')
             {
              chcount =0;
```

```
                ++i;
                --count;
                while( (inp[i] <= '9') && (inp[i] >= '0'))

                 {
                 ch[chcount]=inp[i];
                 ++chcount;
                 ++i;
                 --count;
                 }
                 ch[chcount]='\0';
                 head[j].typ=atoi(ch);
                 head[j].flag=0;
                 head[j].d='n';
                 break;
                } /* end of else */
            else
                {
                 printf("LINE %d:Syntax error,possibly misspelt \"neg / line no.\"at
position %d",j+1,i+1);
                 cflag=1;
                 i+=3;
                 count-=3;
                 err_flag=1;
                 break;
                }

   case 'u':
   case 'U':
          if((( inp[i+1] == 'a' ) || ( inp[i+1] == 'A'))&&
               (( inp[i+2] == 'o' ) || ( inp[i+2] == 'O')))
             {
               i+=3;
               count-=3;
               create(6);
               node->flag=1;
               break;
               }
             else
              if((( inp[i+1] == 'o' ) || ( inp[i+1] == 'O'))&&
                 (( inp[i+2] == 't' ) || ( inp[i+2] == 'T')))
               {
                i+=3;
                count-=3;
                create(7);
                node->flag=1;
                break;
                }
```

```
            else
             if((( inp[i+1] == 'i' ) || ( inp[i+1] == 'I'))&&
               (( inp[i+2] == 'o' ) || ( inp[i+2] == 'O')))
             {
              i+=3;
              count-=3;
              create(8);
              node->flag=1;
              break;
             }
            else
             if(( inp[i+1] == 'r' ) || ( inp[i+1] == 'R'))
             {
              if(( inp[i+2] == 'p' ) || ( inp[i+2] == 'P'))
               {
                i+=3;
                count-=3;
                create(11);
                node->flag=1;
                break;
               }
              else
               if(( inp[i+2] == 't' ) || ( inp[i+2] == 'T'))
                {
                i+=3;
                count-=3;
                create(10);
                node->flag=1;
                break;
                }
             }
        else
          {
           printf("LINE %d:Syntax error,possibly misspelt \"uao / uot / uio / urp /
urt\" at position %d" ,j+1,i+1);
           cflag=1;
           i+=3;
           count-=3;
           err_flag=1;
           break;
             }


case 'M':
case 'm':
          if((( inp[i+1] == 'i' ) || ( inp[i+1] == 'I'))&&
            (( inp[i+2] == 'r' ) || ( inp[i+2] == 'R')))
             {
```

```
                i+=3;
                count-=3;
                create(9);
                node->flag=1;
                break;
                }
          else
           {
            printf("LINE %d:Syntax error,possibly misspelt \"mir\" at position %d"
,j+1,i+1);
            cflag=1;
            i+=3;
            count-=3;
            err_flag=1;
            break;
           }

    case 'D':
    case 'd':
          if(( inp[i+1] == 'i' ) || ( inp[i+1] == 'I'))&&
            (( inp[i+2] == 's' ) || ( inp[i+2] == 'S')))
             {
               i+=3;
               count-=3;
               create(18);
               node->flag=1;
               break;
               }
          else
           {
            printf("LINE %d:Syntax error,possibly misspelt \"dis\" at position %d"
,j+1,i+1);
            cflag=1;
            i+=3;
            count-=3;
            err_flag=1;
            break;
           }

    case 'r':
    case 'R':
           if(( inp[i+1] == 'q' ) || ( inp[i+1] == 'Q'))&&
             (( inp[i+2] == 'o' ) || ( inp[i+2] == 'O')))
             {
               i+=3;
               count-=3;
               create(13);
               node->flag=1;
```

```
                break;
                }
              else
              if((( inp[i+1] == 'p' ) || ( inp[i+1] == 'P'))&&
                 (( inp[i+2] == 't' ) || ( inp[i+2] == 'T')))
              {
                i+=3;
                count-=3;
                ++rcount;
                create(14);
                node->flag=1;
                break;
                }
              else
              {
               printf("LINE %d:Syntax error,possibly misspelt \"rqo(or)rpt\" at position
%d" ,j+1,i+1);
               cflag=1;
               i+=3;
               count-=3;
               err_flag=1;
               break;
              }

  default :

       printf("LINE %d:syntax error at position %d ",j+1,i+1);
       cflag=1;
       --count;
       ++i;
       err_flag=1;
       break;

 } /* switch */
 }
 else
  goto b;
 }
b:  create(40);
node->flag=1;
 node=&head[j];
 if(!err_flag && !flag)
 {
   printf("\n LINE %d IS :%s \n" ,j+1,inp);
 while(node != NULL)
   {
   printf("\t%d ->",node->typ);
```

```
  node=node->next;
  getch();
  }
  printf("\n\n");
}
if(flag)
   printf("\n LINE %d IS END-OF-FILE \n" ,j+1);
  if(!err_flag)
     parser();
   if (j > 0)
    head[j-1].prev = &head[j];
    ++j;
    first_after_g = 0;
    if(flag)
     goto a;
}
a: if(rcount)
   {
    printf("\n Unbalanced RPT - ERP in the program");
    cflag=1;
   }
 if(scount != 0)
   {
      fclose(fptr1);
      fptr1=snode->item;
      spop();
      --scount;
    }
    else
    if( feof(fptr1))
       {
        fclose(fptr1);
        break;
        }
 }
 printf("\n THE SYMBOL TABLE ENTRIES ARE : \n");
 sym=hsym;
while(sym->next != NULL)
{
  if (sym->evar >= 0 && sym->evar <= 19)
                printf("\nevar : %d  & value :%d  \t",sym->evar,sym->value.ival);
             else
               if (sym->evar <= 24)
                printf("\nevar : %d  & value : %ld
\t",sym->evar,sym->value.lival);
             else
               printf("\nevar : %d  & value : %f \t",sym->evar,sym->value.fval);
  sym=sym->next;
```

```
}
if(!cflag)
{
 printf("\n\n\n\t\t********* COMPILATION SUCCESSFULLY COMPLETED
********\n\n\n");
 getch();
 clrscr();
 evalu();
}
else
{
 printf("\n\n\n\t\t ********** COMPILATION ERRORS **********\n\n\n");
 exit(0);
}
/* sym = hsym;
 while (sym != NULL)
  {
  sear = sym->next;
  free(sym);
  sym = sear;
  }
  free(hsym);*/


 j=0;
 while(head[j].prev!=NULL)
 {
  node=head[j].next;
  while(node!=NULL)
  {
      temp=node->next;
      free(node);
      node=temp;
    }
    free(&head[j]);
    ++j;
   }
 getch();
 fflush(stdin);
 return 0;


}
```

```
# include <stdio.h>
# include <alloc.h>
# include <string.h>
# include <stdlib.h>
# include <math.h>
# include <ctype.h>

float cumux,cumuy,cumuz;
float addx,addy,addz;
int inc_mode=0,j;
int erp_flag=0;
FILE  * opfptr;

struct rpt_type
   {
    struct rpt_type * prev;
    int count;
    int r_start;
    struct rpt_type * next;
   };

struct op_type
   {

   struct op_type * prev;
   int op_code;
   int unary;
   struct op_type * next;
   };

struct var_type
    {
    struct var_type * prev;
    int var_or_not;   /* for E-variables , this is set to 1 */
    union
    {
      struct symb *stlo;
      float val;
      }choi;
    struct  var_type *next;
    };

struct rpt_type * rhead, * rnode;
struct var_type * vhead, * vnode;
struct op_type  * ohead,  * onode;
struct tlist  head[50],* node,*te;
struct symb * hsym,*sym;
```

```
struct symb *symlo;
float val;

/* Routine to perform push operation on variable_stack */

void vpush(int fla,float resultant)  /* fla = 1 for an E-variable */
 {

   vnode->next = malloc(sizeof(struct var_type));
   vnode->next->prev = vnode;
   vnode = vnode->next;
   vnode->next = NULL;

   if(fla ==1)
   {
    vnode->var_or_not=1;    /* E-variable is pushed inside variable_stack */

    vnode->choi.stlo=calloc(1,sizeof(struct symb));
    vnode->choi.stlo=symlo;
   }
   else
   {
    vnode->var_or_not=0;
          vnode->choi.val= resultant;

   }

 }

/* Routine to perform pop operation on variable_stack */

void vpop(void)
   {
   vnode->next = NULL;
   vnode = vnode->prev;
   vnode->next->prev = NULL;
   free(vnode->next);
   }

/* Routine to perform push operation on stack,on encountering RPT statement */

void rpush(int count,int start)
  {
   rnode->next = malloc (sizeof(struct rpt_type) );
   rnode->next->prev = rnode;
   rnode = rnode->next;
   rnode->next = NULL;
   rnode->count = count;
```

```
  rnode->r_start = start;

  }

/* Routine to perform pop operation on stack,on encountering ERP statement */

void rpop(void)
 {
  rnode->next = NULL;
  rnode = rnode->prev;
  rnode->next->prev = NULL;
  free(rnode->next);
  }

/* Routine to push operands into the operand_stack */

void opush(int cod)
 {
   onode->next = malloc(sizeof(struct op_type));
   onode->next->prev = onode;
   onode = onode->next;
   onode->op_code=cod;
   onode->next = NULL;
 }

/* Routine to pop operands from the operand_stack */

void opop(void)
  {
  onode->next = NULL;
  onode = onode->prev;
  onode->next->prev = NULL;
  free(onode->next);
  }

/* Routine to perform unary operations */

void u_op( op_code)
{
 if(op_code == 3 || op_code == 4 || op_code ==5)
  {
   if(vnode->var_or_not)
  {
   if (vnode->choi.stlo->evar >= 0 && vnode->choi.stlo->evar <= 19)
              fprintf(opfptr,"%d",vnode->choi.stlo->value.ival);
          else
            if (vnode->choi.stlo->evar <= 24)
              fprintf(opfptr,"%ld",vnode->choi.stlo->value.lival);
```

```
            else
                fprintf(opfptr,"%f",vnode->choi.stlo->value.fval);
    }
    else
     fprintf(opfptr,"%f",vnode->choi.val);
   fputs("   ",opfptr);
   opop();
   return ;
}

 if(vnode->var_or_not)
 {
   vnode->var_or_not=0;
   if (vnode->choi.stlo->evar >= 0 && vnode->choi.stlo->evar <= 19)
                vnode->choi.val =vnode->choi.stlo->value.ival;
             else
               if (vnode->choi.stlo->evar <= 24)
                  vnode->choi.val =vnode->choi.stlo->value.lival;
             else
                  vnode->choi.val = vnode->choi.stlo->value.fval;
    }
   switch(op_code)
   {
      case 24:
           vnode->choi.val=sin(vnode->choi.val * 3.141592 / 180.0);
           break;

      case 25:
          vnode->choi.val=pow(vnode->choi.val,2);
          break;

      case 26:
          vnode->choi.val=cos(vnode->choi.val * 3.141592 / 180.0);
          break;

      case 27:
          vnode->choi.val=tan(vnode->choi.val * 3.141592 / 180.0);
          break;

      case 28:
          vnode->choi.val=asin(vnode->choi.val * 3.141592 / 180.0);
          break;

      case 29:
          vnode->choi.val=acos(vnode->choi.val * 3.141592 / 180.0);
          break;

      case 30:
```

```
            vnode->choi.val=atan(vnode->choi.val * 3.141592 / 180.0);
            break;

        case 31:
            vnode->choi.val=abs(vnode->choi.val);
            break;

        case 32:
            vnode->choi.val=-vnode->choi.val;
            break;

        case 37:
            vnode->choi.val=-vnode->choi.val;
            break;

        default :
                printf("\n INCORRECT UNARY OPERATOR \n");
                exit(0);
      }
    opop();
    return;
   }

/* Routine to perform binary operations */

void b_op(int opcode)
{
  float resul;

  if(opcode == 39)
      {
      if(vnode->var_or_not)
      {
       if (vnode->choi.stlo->evar >= 0 && vnode->choi.stlo->evar <= 19)
                resul =vnode->choi.stlo->value.ival;
              else
                if (vnode->choi.stlo->evar <= 24)
                   resul = vnode->choi.stlo->value.lival;
              else
                   resul =vnode->choi.stlo->value.fval;
          }
          else
            resul = vnode->choi.val;
      vpop();
      if (vnode->choi.stlo->evar >= 0 && vnode->choi.stlo->evar <= 19)
                vnode->choi.stlo->value.ival= (int)resul;
              else
                if (vnode->choi.stlo->evar <= 24)
```

```
                vnode->choi.stlo->value.lival= (int)resul;
            else
                vnode->choi.stlo->value.fval=resul;
    vpop();
    opop();
   }
   else
   {
     if(vnode->var_or_not)

     {
      if (vnode->choi.stlo->evar >= 0 && vnode->choi.stlo->evar <= 19)
              vnode->choi.val = vnode->choi.stlo->value.ival;
            else
              if (vnode->choi.stlo->evar <= 24)
               vnode->choi.val=vnode->choi.stlo->value.lival;
            else
               vnode->choi.val = vnode->choi.stlo->value.fval;
      }

    if(vnode->prev->var_or_not)
     {
         if (vnode->prev->choi.stlo->evar >= 0 &&
vnode->prev->choi.stlo->evar <= 19)
              vnode->prev->choi.val= vnode->prev->choi.stlo->value.ival;
            else
              if (vnode->prev->choi.stlo->evar <= 24)
                 vnode->prev->choi.val=vnode->prev->choi.stlo->value.lival;
            else
                 vnode->prev->choi.val = vnode->prev->choi.stlo->value.fval;
      }

   switch(opcode)
      {
       case 34:
                 printf("\nmultiplying.....");
                 delay(100);
                 resul = vnode->choi.val * vnode->prev->choi.val;
                  printf("\nend of mult %f * %f =
%f",vnode->choi.val,vnode->prev->choi.val,resul);
                 break;

       case 35:
                 resul = vnode->prev->choi.val / vnode->choi.val;
                 break;
       case 36:
                 resul = vnode->prev->choi.val + vnode->choi.val;
                 break;
```

```
        case 37:
                resul = vnode->prev->choi.val - vnode->choi.val;
                break;
        default:
               printf("\n INCORRECT BINARY OPERATOR \n");
              exit(0);
        }
    vpop();
    vpop();
    vpush(0,resul);
    opop();
   }
   return;
}

void inilt(void)
{
    vhead=malloc(sizeof(struct var_type));
    vhead->prev=NULL;
    vhead->var_or_not=0;
    vhead->next=NULL;
    vnode=vhead;

    ohead=malloc(sizeof(struct op_type));
    ohead->prev=NULL;
    ohead->unary=0;
    ohead->next=NULL;
    onode=ohead;
}

/* EVALUATOR : main module */
void evalu()
  {
  char valu[15],c[2];
  char ch;
  char  outfile[11];
   printf("\nEnter output file name  :");
   fflush(stdin);
   gets(outfile);

   if (( opfptr = fopen(outfile,"r")) != NULL )
      {
       printf("\n File already exists \n");
       printf("\n Do you want to overwrite ? (Y/N) : ");
       fflush(stdin);
       scanf("%c",&ch);
       if ( ch == 'Y' || ch == 'y')
          {
```

```
        fclose(opfptr);
        opfptr = fopen( outfile,"w");
        }
     else
       exit(0);
 }
  else
      opfptr = fopen( outfile,"w");


j=0;
 rhead=malloc(sizeof(struct rpt_type));
rhead->prev=NULL;
rhead->count=0;
rhead->r_start=0;
rhead->next=NULL;
rnode=rhead;

while(head[j].prev!= NULL)
{
node=head[j].next;
inilt();
while(node != NULL)
{
if(node->flag)
{
 erp_flag=0;
switch(node->typ)
{
 case 1:
          opush(1);
          onode->unary=0;
          node=node->next;
          break;
  case 2:
          symlo=malloc(sizeof(struct symb));
          symlo=node->stloc;
          vpush(1,0);
          node=node->next;
          break;
  case 3:
          if(node->prev->flag==0)
          {
            if(node->prev->d=='g')
            fputs("\n",opfptr);
           te=node->prev;
           while(te->flag==0 && te !=&head[j])
           {
```

```
            fputc(toupper(te->d),opfptr);
            fprintf(opfptr,"%d",te->typ);
            fputs(" ",opfptr);
            te=te->prev;
          }
        }
        fputc('X',opfptr);
        opush(3);
        onode->unary=1;
        node=node->next;
        break;
case 4:
        if(node->prev->flag==0)
        {
         if(node->prev->d=='g')
         {
           fputs("\n",opfptr);
         }
         te=node->prev;
         while(te->flag==0 && te!=&head[j])
         {
            fputc(toupper(te->d),opfptr);
           fprintf(opfptr,"%d",te->typ);
           fputs(" ",opfptr);
           te=te->prev;
         }
        }
        fputc('Y',opfptr);
        opush(4);
        onode->unary=1;
        node=node->next;
        break;
case 5:
        if(node->prev->flag==0)
        {
         if(node->prev->d=='g')
         {
           fputs("\n",opfptr);
         }
         te=node->prev;
         while(te->flag==0 && te!=&head[j])
         {
            fputc(toupper(te->d),opfptr);
           fprintf(opfptr,"%d",te->typ);
           fputs(" ",opfptr);
           te=te->prev;
         }
        }
```

```
        fputc('Z',opfptr);
        opush(5);
        onode->unary=1;
        node=node->next;
        break;
case 6:
        opop();
        while(node !=NULL)
             node=node->next;
        addx=addy=addz=0;
        cumux=cumuy=cumuz=0;
        inc_mode = 0;
        break;
  case 7:
        node = node->next;
        while(node->typ == 21 || node->typ == 20)
         {
           node = node->next;
         }
         while(node->typ ==  3 || node->typ == 4 || node->typ ==5)
         {
         if (node->typ ==3)
           {
            valu[0]='\0';
            node = node->next;
            while(node->typ == 21 || node->typ == 23)
              {
                c[0]=node->d;
                c[1]='\0';
                strcat(valu,c);
                node = node->next;
                }
            addx=atoi(valu);
           if(node->typ==20)
            node=node->next;
          }

        if (node->typ ==4)
           {
            valu[0]='\0';
            node = node->next;
            while(node->typ == 21 || node->typ == 23)
              {
                c[0]=node->d;
                c[1]='\0';
                strcat(valu,c);
                node = node->next;
                }
```

```
            addy=atoi(valu);
           if(node->typ==20)
            node=node->next;
          }
       if (node->typ ==5)
          {
           valu[0]='\0';
           node = node->next;
           while(node->typ == 21 || node->typ == 23)
              {
                c[0]=node->d;
                c[1]='\0';
                strcat(valu,c);
                node = node->next;
                }
           addz=atoi(valu);
          if(node->typ==20)
           node=node->next;
         }
        }
       break;
case 8:
     inc_mode = 1;
     node = node->next;
       while(node->typ == 20)
        {
          node = node->next;
        }
        while(node->typ ==  3 || node->typ == 4 || node->typ ==5)
        {
        if (node->typ ==3)
          {
           valu[0]='\0';
           node = node->next;
           while(node->typ == 21 || node->typ == 23)
              {
                c[0]=node->d;
                c[1]='\0';
                strcat(valu,c);
                node = node->next;
                }
           cumux=atoi(valu);
          if(node->typ==20)
           node=node->next;
         }

       if (node->typ ==4)
          {
```

```
            valu[0]='\0';
            node = node->next;
            while(node->typ == 21 || node->typ == 23)
               {
                c[0]=node->d;
                 c[1]='\0';
                 strcat(valu,c);
                 node = node->next;
                 }
            cumuy=atoi(valu);
           if(node->typ==20)
            node=node->next;
         }
      if (node->typ ==5)
          {
           valu[0]='\0';
           node = node->next;
           while(node->typ == 21 || node->typ == 23)
              {
                c[0]=node->d;
                c[1]='\0';
                strcat(valu,c);
                node = node->next;
                }
           cumuz=atoi(valu);
           if(node->typ==20)
            node=node->next;
         }
        }
       break;

case 9:


case 10:
case 11:
case 12:
case 13:
      while(node != NULL)
        node = node->next;
     break;

case 14:
      node = node->next;
      node = node->next;
      valu[0] = '\0';
      while(node->typ == 21)
              {
```

```
                c[0]=node->d;
                c[1]='\0';
                strcat(valu,c);
                node = node->next;
                }

        rpush( atoi(valu) , j+1);
        node = node->next;
        node=node->next;
        break;

case 15:
        --rnode->count;
        if (rnode->count ==0)
          {
           erp_flag =0;
           rpop();

          }
         else
          {
          erp_flag = 1;
          j = rnode->r_start;
          }
         while (node != NULL)
         node = node->next;
         break;

case 16:
   /* subroutines already processed */
case 17:
 case 18:
          while(node!=NULL)
            node = node->next;
         break;

 case 21:

        valu[0] = '\0';
        while (node->typ == 21 || node->typ == 23)
            {
             c[0]=node->d;
             c[1]='\0';
             strcat(valu,c);
             node = node->next;
            }
        val=atof(valu);
        vpush(0,val);
```

```
        break;

    case 24:
            opush(24);
            onode->unary = 1;
            node = node->next;
            break;

    case 25:
            opush(25);
            onode->unary = 1;
            node = node->next;
            break;

    case 26:
            opush(26);
            onode->unary = 1;
            node = node->next;
            break;

    case 27:
            opush(27);
            onode->unary = 1;
            node = node->next;
            break;

    case 28:
            opush(28);
            onode->unary = 1;
            node = node->next;
            break;

    case 29:
            opush(29);
            onode->unary = 1;
            node = node->next;
            break;

    case 30:
            opush(30);
            onode->unary = 1;
            node = node->next;
            break;

    case 31:
            opush(31);
            onode->unary = 1;
            node = node->next;
```

```
            break;

        case 32:
            opush(32);
            onode->unary = 1;
            node = node->next;
            break;

        case 33:
            opush(33);
            onode->unary = 1;
            node = node->next;
            break;

        case 34:
                while(onode->unary==1 && (onode->op_code !=3 &&
onode->op_code !=4 && onode->op_code != 5))
                {
                 u_op(onode->op_code);
                }
            opush(34);
            onode->unary =0;
            node = node->next;
            break;

        case 35:
            while (onode->unary == 1 && (onode->op_code !=3 &&
onode->op_code !=4 && onode->op_code != 5))
                {
                 u_op(onode->op_code);
                }
            opush(35);
            onode->unary =0;
            node = node->next;
            break;

        case 36:
            while((onode->unary == 1 && (onode->op_code !=3 &&
onode->op_code !=4 && onode->op_code != 5))
                    || onode->op_code == 34 || onode->op_code == 35)
            {
             if(onode->unary == 0)
             {
             b_op(onode->op_code);
             }
             else
             {
              u_op(onode->op_code);
```

```
        }
        }
        opush(36);
        onode->unary =0;
        node = node->next;
        break;

    case 37:
        while((onode->unary == 1 && (onode->op_code !=3 &&
onode->op_code !=4 && onode->op_code != 5))
        || onode->op_code == 34 || onode->op_code == 35)
        {
        if(onode->unary == 0)
        {
        b_op(onode->op_code);
        }
        else
        {
        u_op(onode->op_code);
        }
        }
        opush(37);
        if ( (node->prev->typ == 39)||(node->prev->typ == 1))
            onode->unary =1;
        else
            onode->unary =0;
        node = node->next;
        break;

    case 38:
        while(onode->op_code != 1)
        {
        if(onode->unary == 0)
        {
        b_op(onode->op_code);
        }
        else
        {
        u_op(onode->op_code);
        }
        }
        opop();
        node=node->next;
        break;
    case 39:
        opush(39);
        onode->unary=0;
        node=node->next;
```

```
            break;

      case 40:
          while(onode != ohead)
           {
            if(onode->unary ==  0)
            {
               b_op(onode->op_code);
             }
            else
             {
               u_op(onode->op_code);

             }
           }
           node=node->next;
           break;
        default :printf("\n UNKOWN TOKENS \n");
                 break;
       }
      }
     else
      node=node->next;
      }
      if(!erp_flag)
      ++j;
      sym=hsym;
      printf("\n\n");
while(sym->next != NULL)
{
   if (sym->evar >= 0 && sym->evar <= 19)
               printf("\nevar : %d & value :%d  \t",sym->evar,sym->value.ival);
            else
              if (sym->evar <= 24)
               printf("\nevar  :  %d      &  value  :%ld
\t",sym->evar,sym->value.lival);
            else
              printf("\nevar : %d & value :%f  \t",sym->evar,sym->value.fval);
    sym=sym->next;
  }
    }

    fclose(opfptr);
    printf("\n\n\n\t\t FILE CLOSED \n\n\n");
    return;
   }

}
```

```
# include <stdio.h>
# include "symbol.h"
void main()
 {
  struct type_ele element;
  int ch=1;
  int row=1,column=1,prncol=2924;
  FILE *fptr2;
  while(ch ==1)
  {
  row=1;
  column=1;
  fptr2=fopen("table5.dat","ab");
/*   fseek(fptr2,sizeof(struct type_ele)*1836,SEEK_SET);*/
  printf("\nEnter the elements\n");
  while(row <= 1 )
  {
  while(column <= 68 )
    {
   if ((column >=1 && column <=2) || (column >=6 && column <=19) ||
     (column >= 22 && column <= 37) || (column >=39 && column <=55)
||
     (column >= 59 && column <=62) || (column >=64) )
     {
      element.c = 'e';
      element.i = 1;

      }
    else
    {
      printf("\nEnter the element in row %d column %d",row,column);
      fflush(stdin);
      scanf("%c%d",&element.c,&element.i);
     }
      fwrite(&element,sizeof(struct type_ele),1,fptr2);
      ++column;
      ++prncol;
    }
    ++row;
  }
     fclose(fptr2);


  fptr2=fopen("table5.dat","rb");
  fseek(fptr2,sizeof(struct type_ele)*2652,SEEK_SET);
  row=1;column=2653;
   while(row <= 1 )
  {
```

```
  while(column <= prncol )
    {
      fread(&element,sizeof(struct type_ele),1,fptr2);
         printf("\nThe element in row %d col %d is
%c%d",row,column,element.c,element.i);
      ++column;
    }
     ++row;
  }

  fclose(fptr2);

printf("\n Continue ? (1/2)");
scanf("%d",&ch);
}
}
```

```
# include <stdio.h>
# include "symbol.h"
void main()
  {
   struct type_prod prodval,prod[98];
   int co=0;
   FILE *fptr3;
/*   fptr3=fopen("prodc.dat","r+w");
   printf("\nEnter the elements\n");
   while(co < 98 )
   {
       printf("\nEnter the value of %d element :",co+1);
       fflush(stdin);
       scanf("%d",&prodval.c);
       printf("\nEnter the beta of %d element :",co+1);
       fflush(stdin);
       scanf("%d",&prodval.beta);
       fwrite(&prodval,sizeof(struct type_prod),1,fptr3);
       ++co;

   }
   fclose(fptr3);*/
   co =0;
   fptr3=fopen("prodc.dat","r");

   printf("\n PRINTING VALUES \n");

   while(co < 98)
     {
       fread(&prod[co],sizeof(struct type_prod),1,fptr3);
       printf("\nValue %d is : %d",co+1,prod[co].c);
       printf("\nBeta  %d is : %d",co+1,prod[co].beta);
       ++co;
     }
 fclose(fptr3);
 }
```

BIBLIOGRAPHY

- PRINCIPLES OF COMPILER DESIGN
  - ALFRED. V. AHO
  - JEFFREY. D. ULLMAN

- COMPILER DESIGN IN C
  - ALLEN. I. HOLUB

- SOFTWARE ENGINEERING
  A PRACTITIONER'S APPROACH, THIRD EDITION
  - ROGER. S. PRESSMAN

- 8600 MC PROGRAMMING & OPERATORS MANUAL