Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

1984-06

An input translator for a computer-aided design system

Carson, Thomas H.

https://hdl.handle.net/10945/19386

Downloaded from NPS Archive: Calhoun

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN INPUT TRANSLATOR FOR A
COMPUTER-AIDED DESIGN SYSTEM

by

Thomas H. Carson

June 1984

Thesis Advisor: Alan A. Ross

Approved for public release; distribution unlimited

T217387

DUDLEY KNOX LIBRARY
NAVAL POSTG...
MONTEREY, ...

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): An Input Translator for a Computer-Aided Design System	5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis, June, 1984 / 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Thomas H. Carson	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93943	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93943	12. REPORT DATE: June, 1984 / 13. NUMBER OF PAGES: 120
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED / 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer-Aided Design, Input Translator, CSDL (Computer System Design Language), CSDE (Computer System Design Environment)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The purpose of this thesis is to design and implement the input translator for the Computer System Design Environment, which is a computer-aided design system. The Computer System Design Environment is used to design real time controllers for a variety of purposes. The input translator will take an input, which has been developed in the prescribed language, CSDL, and with the aid of a partial syntax-directed editor, (Continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

ABSTRACT (Continued)

translate it into primitive list form. This form is used by the remainder of the system to select the best hardware and software components, as described in a set of realization libraries, to build the proposed controller.

Approved for public release; distribution unlimited.

An Input Translator for a Computer-Aided Design System

by

Thomas H. Carson
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 1971

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1984

ABSTRACT

The purpose of this thesis is to design and implement the input translator for the Computer System Design Environment, which is a computer-aided design system. The Computer System Design Environment is used to design real time controllers for a variety of purposes. The input translator will take an input, which has been developed in the prescribed language, CSDL, and with the aid of a partial syntax-directed editor, translate it into primitive list form. This form is used by the remainder of the system to select the best hardware and software components, as described in a set of realization libraries, to build the proposed controller.

TABLE OF CONTENTS

I. INTRODUCTION AND BACKGROUND 8
A. COMPUTER-AIDED DESIGN 8
B. COMPUTER SYSTEM DESIGN ENVIRONMENT 9
C. PREVIOUS WORK 10
D. SCOPE OF THIS THESIS 12

II. TRANSLATOR DESIGN 13
A. DESIGN REQUIREMENTS AND CONSTRAINTS 13
B. CSDL- THE INPUT LANGUAGE 14
C. PARSER ALTERNATIVES 17
D. PARSER STRUCTURE 19

III. TRANSLATOR IMPLEMENTATION 21
A. LANGUAGE OF IMPLEMENTATION 21
B. TRANSLATOR INPUT 22
C. PRINCIPAL PROCEDURES AND DATA STRUCTURES . . . 23
D. TRANSLATOR OUTPUT 26
1. Primitive List 26
2. Symbol Table 27
3. Scratch Pad 28

IV. TESTING AND VALIDATION 30
A. THEORY OF TESTING 30
B. TEST RESULTS 31

V. CONCLUSIONS AND RECOMMENDATIONS 38
A. PROGRAM MAINTENANCE 38
B. RECOMMENDATIONS FOR CSDE 39
C. SUMMARY 40

APPENDIX A: TRANSLATOR INPUT EXAMPLE 41

APPENDIX B: FORMAL SYNTAX OF CSDL 44

APPENDIX C: PRIMITIVE LIST 55

APPENDIX D: TRANSLATOR SOURCE LISTING 59

APPENDIX E: SCRATCH PAD EXAMPLE 110

APPENDIX F: SYMBOL TABLE 116

LIST OF REFERENCES 118

INITIAL DISTRIBUTION LIST 120

LIST OF FIGURES

1.1 Computer System Design Environment 11
2.1 A Partial Example of CSDL 14
2.2 An Example of CSDL Syntax 15
2.3 Parser Structure 19
3.1 A Sample of the Primitive List 26
4.1 IDENTIFICATION Section of the Input 32
4.2 DESIGN CRITERIA Section of the Input 32
4.3 Primitive List Form of the DESIGN CRITERIA . . . 33
4.4 ENVIRONMENT Section of the Input 33
4.5 Primitive List Form of the Input 34
4.6 PROCEDURES Section Input Example 35
4.7 Primitive List Form of the PROCEDURES Section 36
4.8 CONTINGENCY LIST Input Example 36
4.9 Primitive List Form of the CONTINGENCY LIST . . 37

I. INTRODUCTION AND BACKGROUND

A. COMPUTER-AIDED DESIGN

Computers, as design tools, are beginning to touch every facet of our lives. We can turn on our television sets to see an advertisement for an automobile with a computer being used to aerodynamically design the body style of the car. Architects are using computers to assist in modern drafting and architectural design techniques [Ref. 1]. The potential for automating the design of many other products exists and research in all areas of computer-aided design (CAD) is continuing at a prolific rate.

While research in the areas of artificial intelligence may lead us someday to a computer which can, using natural language understanding, solve problems reserved for only humans today, current technology limits us to those problems where the computer relies on human expert input for a knowledge base. A knowledge base developed by human experts is used by the computer to derive a design much faster than a human and reduce the concern over the complexity of the process. The computer can maintain a large data base of components from which it can pick to satisfy a criteria as described by the designer. While not really creative, this system allows the mixing and matching, automatically, of components to produce the best combination available. The advantages to be gained are a decrease in the time it takes to complete the design process and error free results, while leaving the human designer to concentrate on the desired specification.

One of the most important features of such a system is its accessibility. The user interface must be one that meets the needs of the designer while remaining within the bounds of current technology. The interface must be user-friendly to the greatest possible extent. It is this particular portion of the problem which has given rise to the most debate and brought forth the widest range of possible solutions.

B. COMPUTER SYSTEM DESIGN ENVIRONMENT

The Computer System Design Environment (CSDE), under development at the Naval Postgraduate School in Monterey California, is one such computer-aided design system. It is based on the research contained in LtCol. Alan Ross's doctoral dissertation [Ref. 2]. Ross's work is an expansion and realization of the research conducted by M. N. Matelan [Ref. 3]. The first CSDE implementation is one in which real-time controllers (microprocessers) are designed based on a realization library (knowledge base) of current microprocesser technology. The system creates the problem statement in a syntax-directed editor, translates it into an intermediate form, selects a microprocessor realization from the library and generates the software and hardware component descriptions to implement the design. The components are used to select a processor volume or implementation. The volume checked to see if the timing constraints, set forth by the designer, can be achieved. If so, the monitor is generated and the output is formatted. The monitor produces the software and ancillary hardware to realize the correct strategy. If the timing constraints cannot be met, a new volume must be chosen and tested. The CSDE gives a designer the tools to derive the appropriate components that make up the controller, no matter what its task is to be.

Motivation and discussion of the CSDE are contained in [Refs. 2 , 3].

C. PREVIOUS WORK

The modules that make up the Computer System Design Environment are depicted in figure 1.1. Matelan described a Control System Design Language (CSDL) as the input language for this system [Ref. 3]. Using CSDL with a syntax-directed editor keeps the input details at a high-level of abstraction while completely describing the proposed design.

To fulfill the input requirements of block 1 in figure 1.1, a syntax-directed editor was designed and partially implemented as a result of Lt. Barbara Sherlock's thesis at the Naval Postgraduate School in 1983 [Ref. 4]. Lt. Sherlock's editor receives a high level input description of the problem from the designer, formats it and passes it to the input translator. This form is a combination of Matelin's CSDL and ADA, the Department of Defense sponsored design language. This language, as the basis for input to and output from the editor, follows the concept that the problem statement should not require the designer to be proficient in the details of a high level programming language. The translator, as its name suggests, translates the output from the editor into an intermediate form acceptable to the follow on CSDE processes. Its design and implementation are the subjects of this thesis. The optimizer and functional mapper(Blocks 3 and 4, Figure 1.1) exist as Fortran programs in the CSDE. The optimizer requires an 80 column format for its input which is a primitive list or the set of functions that the controller will perform, in an almost assembly-like language format. It is developed from the contingency and procedures sections of the design input statement. Hardware availability

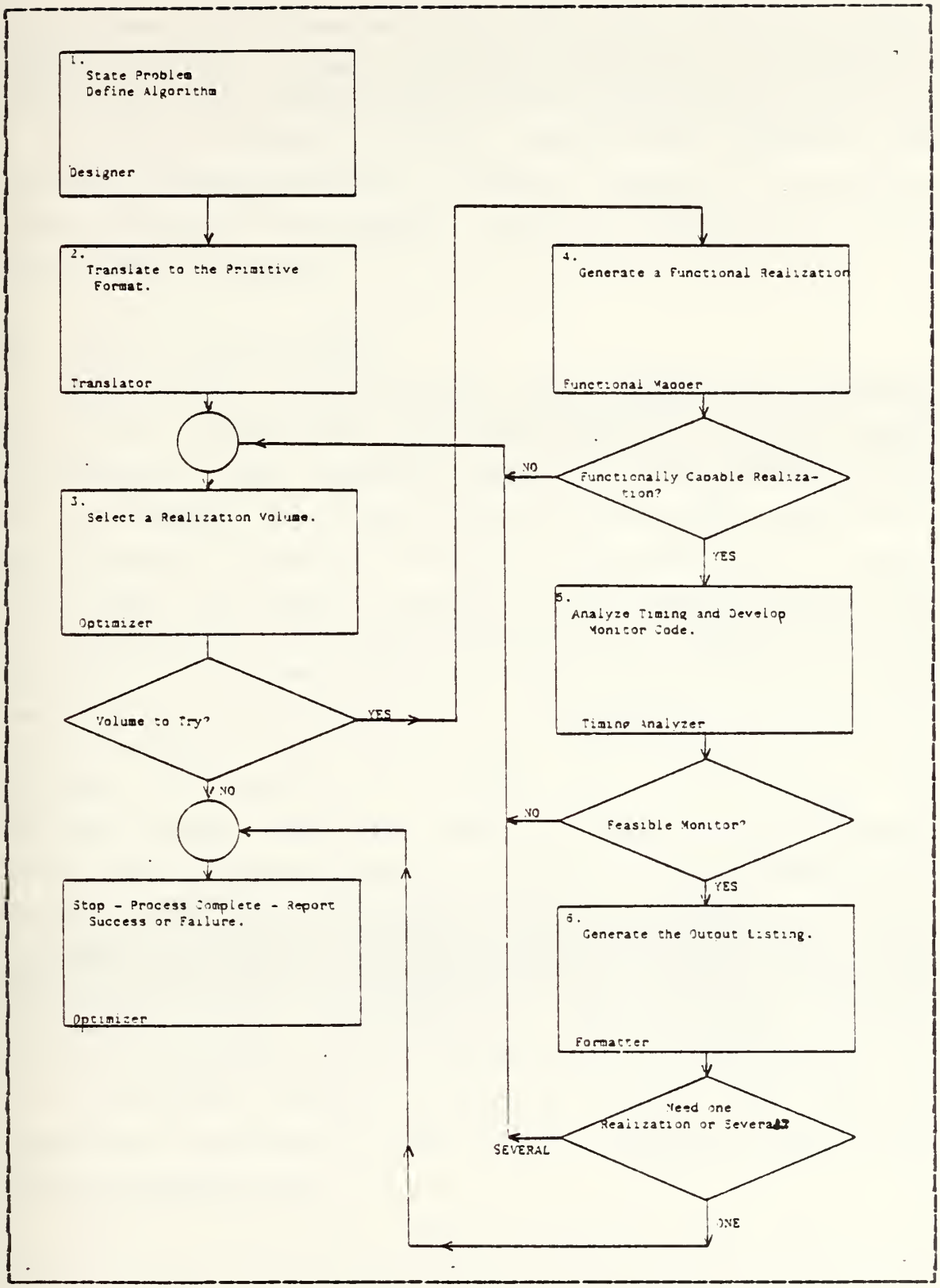


Figure 1.1 Computer System Design Environment.

prescribed this format when the CSDE was originally implemented by LtCol. Ross. Work is currently in progress to install both the design input and the realization library in a relational database, thereby updating the optimizing and mapping processes [Ref. 5]. But the basis of the input to these sections of the system, which is the primitive list, will remain the same.

D. SCOPE OF THIS THESIS

The purpose of this thesis is to design and implement the input translator for the Computer System Design Environment. The translator must take the problem statement, a design for a real time microprocessor controller, and translate it into a list of primitives and a symbol table which can then be mapped across a realization library to determine the most feasible components. Appendix A is an example of a problem statement in the Computer System Design Language developed by Matelan. In addition, the translator must produce a timing table which can be used by the timing analyser to determine a feasible monitor to control the complete device [Ref. 2]. The timing table is developed using the contingency section of the problem statement which contains the timing requirements for the problem.

The next chapter of this thesis describes the design considerations for the translator and provides the detailed discussion of its requirements. The following chapters discuss the implementation and testing of the translator and the conclusions reached during the work. In addition, Appendice D includes the code for the translator with appropriate documentation.

II. TRANSLATOR DESIGN

A. DESIGN REQUIREMENTS AND CONSTRAINTS

As previously discussed, the input translator can be thought of as one of several modules in the CSDE system. In the CSDE hierarchy, it lies between the designer and the optimizer. If we consider, for a moment, the module as a black box, then we can better describe its function. The input to the module is a specification, written by the designer, for a real time controller. While Lt. Sherlock, in her design of the input editor [Ref. 4], decided to produce a pseudo-ADA specification language as the output of the editor, the translator being designed by this author will use the Control System Design Language. If the ADA ouput was to be adapted, this language would have to be formalized and a grammar produced that is capable of being parsed. In addition, the pseudo-ADA provides no real advantage, as the specification must be parsed and the same output produced no matter what the language. Therefore, with the additional knowledge that the editor is the subject of a current thesis project which will return to the CSDL output, the decision was made to write the translator for that CSDL. A partial example of the CSDL description is contained in figure 2.1.

The output from the translator consists of a primitive list, a symbol table, and an application timing table. The primitive list is intermediate code which reflects the requirements of the input while the symbol table contains all input variables and their attributes. The application timing table contains the contingencies with their related tasks and all supplied timing values from the problem definition. This table is used during the timing analysis.

```
CONTINGENCY LIST
  WHEN ALARM : 2MS,50US DO ALERT;
  EVERY 4MS : DO ENCODE;
  WHEN DATA_READY : 1300US DO SERIALIZE;
PROCEDURES
 FUNCTION DATA_READY:
  BINARY,1;
  SENSE (BUFFER);
  IF BUFFER /= OLDBUF THEN DATA_READY := 1 END IF;
 EXIT DATA_READY;
```

Figure 2.1 A Partial Example of CSDL.

The requirements, then, exist for the input language, CSDL, to be analyzed to determine what method of translation is to be employed. In addition, the required output must be standardized among the system modules so the proper semantics can be developed for the translation process. Each of these issues will be discussed in detail in tne following sections.

B. CSDL- THE INPUT LANGUAGE

A translator accepts a source program, written in a source language, and transforms it into an object program [Ref. 6]. A source language designed for use in a computer aided design system and utilized in CSDE is the Control System Design Language (CSDL), the origin of which was previously discussed. It is composed of an alphabet whose individual elements are called tokens and a grammar which expresses the rules governing the legal classes of token strings. The tokens can be futher subdivided into terminals

and nonterminals. Terminals are the letters of the allowed alphabet while the nonterminals are representations of strings in the language which increase the expressive power. A partial example of the production rules for CSDL is contained in figure 2.2. The syntax for CSDL, which includes the alphabet and grammar, is contained in Appendix B.

```
<WHEN DO> ::= <QUALIFICATION> WHEN <NAME>
              <EPISODE TIMING> DO <TASK LIST>

<TASK LIST> ::= <NAME> / <TASK LIST> THEN <NAME>

<NAME> ::= *ID* / *ID* ( <EXPR LIST> ) /
           *ID* *NUMBER* : *NUMBER*
```

Figure 2.2 An Example of CSDL Syntax.

Different types of translating devices accept different languages classifications. To narrow the choice of translator designs, it must be determined which classification fits CSDL. Neither Matelan nor Ross, in their early work on CSDE, included this description [Refs. 2 , 3]. So, a brief review of language classification will assist in this determination. Chomsky distinguished four general classes of grammars [Ref. 7]. Without turning this section into a text on language theory, these are, from the most general to the most specific: unrestricted, context-sensitive, context-free, and right-linear. Context-sensitive and context-free are subsets of the unrestricted class, while the right-linear and two other grammars related to the right-linear

grammars, left-linear and regular, are all subsets of the context-free grammars. These classes allow us to define sentence recognizing machines which form the basis for translators. CSDL falls into the category of context-free grammars. This is the set which, in its production rules, has any string of terminals and nonterminals on the right-hand side of the production while the left-hand side is restricted to nonterminals only. The classes of right-linear, left-linear, and regular grammars restrict the order and appearances of terminals and nonterminals on each side of the production rules and CSDL does not fall into one of these categories. Context-sensitive grammars allow terminals as well as one nonterminal on the left-hand side of the production rules and,while CSDL does fit this category, the context-sensitive are a super-set of the context-free grammars, so this is not an issue when we try to develop the machines which can recognize CSDL.

Each of the phrase-structured grammar classes has an automaton associated with it. The right-linear grammars can be recognized and accepted by a finite-state automaton which consists of a finite set of states and a set of transitions between pairs of states. Each transition is associated with some terminal symbol. The context-sensitve grammars are recognized and accepted by a two-way, linear bounded automaton which is essentially a Turing machine whose tape cannot grow longer than the input string. And, finally, context-free grammars are recognized and accepted by a finite-state automaton controlling a push-down stack, with rules governing the operations on the stack [Ref. 6].

Matelan states that CSDL was created as a context-free grammar and inspection of the syntax contained in Appendix B confirms this [Ref. 3]. In order to recognize strings in the language and translate them into the presribed primitive-list format a finite-state automaton with a push-down stack will be developed.

C. PARSER ALTERNATIVES

It was shown above that CSDL is a context-free grammar and a push-down automata will be required as the recognizer for strings in the language. Additional properties of CSDL must be investigated to further define the problem of parsing. In a context-free grammar each nonterminal can be expanded into some terminal string independently of its neighbors, and its expanded string essentially "pushes aside" its neighbors without interfering with their order in any way [Ref. 6]. But we do impose some ordering rule for the selection of the next nonterminal to replace, in a sentential form for a canonical derivation. The most common rules are left-most and right-most. In a left-most derivation, the left-most nonterminal in each sentential form is selected for the next replacement and in a right-most, the right-most nonterminal is selected. Most common programming languages are easily parsed from left to right, but with difficulty from right to left. Futhermore, algebraic operations are usually performed from left to right, by convention, so it is the order that will be considered. A top-down parse of some sentence, scanning from left-to-right through the string, corresponds to a left-most derivation while a bottom-up parse works from a given sentence upward toward the start symbol, in a left-to-right manner. [Ref. 6].

There are some rules governing the use of these two types of parsers which affect the choice of one for use in recognizing CSDL. Top-down recognition with a look-ahead of k symbols is only possible on a subset of the context-free grammars called LL(k) grammars. Although it is not obvious whether a grammar is LL(k), there is one property which is relevant in this discussion. An LL(k) grammar has no left recursive nonterminals,i.e., a nonterminal A, such that A =>

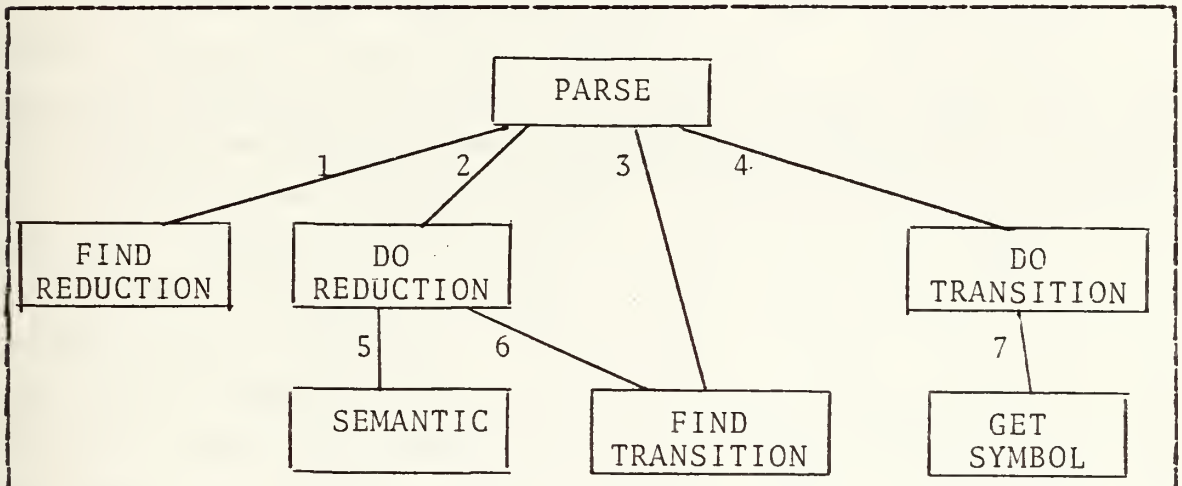
Aw for some w, a string in the language [Ref. 6]. It can be quickly determined by examining Production 17 in Appendix B that CSDL is left-recursive. In fact, it is full of recursion. There are algorithms for removing left-recursion in grammars and for a small grammar that would be the choice. But CSDL has 190 production rules and removing the extensive recursion would increase the grammar size an unacceptable amount. So top-down parsing will be discarded as a possible parsing method.

A bottom-up LR(k) parser is the other major type of recognizer under consideration. A grammar is said to be LR(k) if, for every derivation, the production A => x can be inferred by scanning ux and at most the first k symbols of v in the following derivation step: uAv => uxv . The major advantage of this method is that an LR(k) parser can be constructed for any context-free grammar. This would eliminate the necessity to remove the left-recursion from CSDL.

There is one other major advantage in choosing a bottom-up LR(1) parser automaton. An automatic parser generator can construct, using a computer, the language specific tables that control the operation of the automaton. The LR package from Lawrence Livermore Laboratory [Refs. 8 , 9] is such a system which constructs the tables. Having it available on the Vax 11/780 at the Naval Postgraduate School made the decision easy. It,also, has the advantages that the parsing routine is guaranteed to be correct, the CSDL grammar can be changed easily when necessary, and the resulting translator becomes simple and efficient. For the details as to how the package works see References 7 and 8.

D. PARSER STRUCTURE

The structure of the parser will follow the technique described by G.J. Myers in his book, Composite Structured Design [Ref. 10], and utilized in the design and implementation of an ADA pseudo-machine by Captain Alan Garlington in his thesis [Ref. 11]. The hierarchy for such a technique is



	IN	OUT
1	Current state and token	Production #
2	Production #, Stkptr	New state #, Stkptr
3	Current state and token	New state #
4	New state #, Stkptr	Stkptr, Current state Token description
5	Production #, Stkptr	----------
6	Stack(stkptr) state, Left-hand side (production #)	New state #
7	----------	Token description

Figure 2.3 Parser Structure.

depicted in figure 2.3. The top module, PARSE, provides FINDREDUCTION with the current state and current look-ahead symbol. FINDREDUCTION returns a production number if any

reduction exists. PARSE then calls DOREDUCTION and the sequence is repeated. If no reduction exists, PARSE calls FINDTRANSITION to see if any transitions exist. DOTRANSITION accomplishes the transition and the routine repeats until a final state is reached. If no transition exists, an error is detected and the routine either attempts to recover or halts depending on the severity of the error.

The parser maintains two stacks, one to store the next token and one to store the current state. When DOREDUCTION provides SEMANTIC with the production number, the components of the production have been placed on the stack and SEMANTIC can take the the proper action. This action could include adding a symbol to the symbol table with its appropriate parameters, calling an error routine,or nothing. After completing the semantic actions the items on the stack are removed by DOREDUCTION and the proper token replaces them. Various auxiliary procedures and error routines will be necessary to complete the translator. These procedures will include the input and output routines required to read the designers input and place the created primitives and symbol table in proper format.

The translator, thus, will take a CSDL description of the desired controller, check for errors, parse the input, and produce the primitive list , the symbol table, and application timing table. The implementation of the translator is discussed in the following chapter.

III. TRANSLATOR IMPLEMENTATION

A. LANGUAGE OF IMPLEMENTATION

The Computer System Design Environment was originally implemented in Fortran, with the system maintaining its data base on formatted punch cards for use in a batch environment. Feasible alternatives exist and have been investigated in subsequent research. CSDE is now installed in the Vax 11/780 at the Naval Postgraduate School, with interactive computing available in a variety of programming languages. However, it is still maintained, mainly, in Fortran. It has been shown to be conceptually feasible to represent the data base requirements of the CSDE by a relational model [Ref. 5]. Future intentions are to realize this concept on a data base management system such as Oracle, which is available on the Vax machine. The importance of this is that format, such as column numbers and location of data, and interface compatibility lose their importance and the relations between the data and the representations of the relations become major concerns.

An additional property required for the implementation language is maintainability. CSDL will not be static, as previously discussed. As hardware technology changes, the realization library will have to be updated and, accordingly, CSDL will have to modified. This will result in an update to the translator, reflecting, possibly, such changes as new primitives. A high level familiar programming language will ease the burden of maintenance for future users.

Pascal is a high level programming language which supports the above requirements and, therefore, was chosen over the possible alternatives. Pascal is familiar to most programmers and,in fact, is the "first language" taught to new Computer Science students at the Naval Postgraduate School. In addition, it is easily understood by programmers conversant with other block-structured languages.

The modularity available with Pascal's procedures, functions, and high level constructs will provide maintainability. Each major function in the parser will comprise a Pascal procedure, making the main body of the program simple. Also, a section to be modified or updated is self-contained, and can be seperately compiled and debugged following changes.

Perhaps the most important reason for selecting Pascal is the expected improvements in the CSDE system data base. When the libraries are placed in a relational database, and the designer's input must be mapped to a library in such a system, the data structures comprising the translator's outputs will require change. Pascal data structures are powerful and adaptable to a relational model, enabling this major modification to be completed without difficulty. Until such time as this occurs, the output will be of the form dictated by current system implementation. The details of this will follow.

B. TRANSLATOR INPUT

The input to the translator is the designer's requirements for a controller, written in the Control System Design Language, an example of which is contained in Appendix A. At this writing, a partial syntax-directed editor is under development which will create, based on the designer's ideas, the input in syntactically correct form. No

conceptual basis for the storage or file format of the input has been previously discussed, so several assumptions were made to enable production of the translator. Since the program is to be written in Pascal and reside as a member of the larger system, the CSDE, a simple file containing the CSDL problem description, which is read by the translator, will be utilized as the method of input. The file will be a text file with the only formatting restrictions being those imposed by the syntax of CSDL, the language being parsed. While this method of input will, currently, require some hands on system manipulation during run time, it is envisioned that a system macro can easily be developed at a later date to automate the process.

It is intended that the input be provided to the translator in syntactically correct form. However, as mentioned above, the means for this is not yet implemented. As example problem statements and test cases have been generated to exercise the translator, a requirement has developed for syntax error detection. This requirement can be eliminated and the ensuing code removed upon completion of the editor.

C. PRINCIPAL PROCEDURES AND DATA STRUCTURES

The parser was summarized at a high level of abstraction in Chapter 2 of this thesis. This section will point out the important procedures and data structures employed by the translator. The supporting functions which complete the translator are described as they appear in the program in Appendix D.

The tables produced by the automatic parser generator, which control the operation of the parser, are placed in arrays. The array sizes were set in program constants and would be modified if a change to CSDL caused a modification

to the tables. In addition, the symbol table, the state and lookahead stacks, and the temporary and constant lists are all implemented as arrays of records. Each record contains such information as the type and precision of the variable or constant and a pointer to the next record in the list. With the continuous manipulation of these data structures, such as pops and pushes on the stacks as well as the requirement for access to each member of each list, it was determined that this implementation allowed the maximum degree of flexibility. Also, the size of each data structure was described by a program constant, in order to improve maintenance. A limit might change in a situation where a controller design required a large number of input and output signals or internal variables, exceeding the maximum allowed for a stack.

Four sections conceptually comprise the translator program. The first is the initialization sequence, comprised of the procedure INITIALIZE and supporting functions. This section sets the initial values for all program variables and initializes the temporary and constant lists as well as the input symbol table to null values. It also establishes the SYMTABLE, which is a list containing all reserved words in CSDL, each located by a pointer. This table is used in the program to check each input token to determine whether it is a reserved word, identifier, or operator. Additionally, INITIALIZE includes the procedures for sending the generated primitive list, symbol table, and scratch pad to output files.

The next section is the actual parsing routine, comprised of the procedures PARSE, FINDREDUCTION, DOREDUCTION, FINDTRANSITION, DOTRANSITION, and their supporting procedures and functions. This sequence of code, essentially, repeats itself, looking at each input token, retrieved by GETSYM, and attempts to move through the

required production until a final state is reached. The tokens are placed on the stack and if a reduction can be performed, control moves to the semantic portion of the program. If one cannot be done, the sequence finds the transition and continues movement through the production rule, getting the next input symbol, with the stack unchanged. If no transitions exist, an error is present in the input.

When a reduction number, which corresponds to a production number in the CSDL grammar, is found, the program sequences to the third main section which contains the semantic operations for the program. It includes the procedures SEMANTIC and SEMANTIC1. These are two large case statements which, for each production, do the proper stack operations and send the output information to the procedures in the initialization section to be formatted and filed. The semantic operations are called from within DOREDUCTION, but really comprise a seperate module within the structure of the translator.

The last section of the translator is the set of procedures comprising the error handling routines. This includes procedures PRINTERRORS, RECOVER, ERROR, and PRINTLINERRORS. The parsing routine attempts to recover, using the procedure of the same name, from an error while documenting it to the user. If the parser cannot recover, the program will halt and print the complete list, in a file, of the errors noted prior to the crash. This error handling sequence can be eliminated from the translator, as previously noted, on completion of a syntax directed editor, which would ensure error-free input.

D. TRANSLATOR OUTPUT

1. Primitive List

The primary output of the translator is the primitive list, a sample of which is contained in figure 3.1. This list of primitives is similar to a set of macros with

```
P   1s.generated for:DATAA
P   2s.proc      (DATAA:)
P   3s.sensecond (FLGA:1)
P   4s.eq        (@T01,FLGA,@C01:8,1,8)
P   5s.jmpf      (@T01,@01:8)
P   6s.assign    (DATAA,@C01:1,8)
P   7s.loc       (@01:)
P   8s.exitproc  (DATAA:)
```

Figure 3.1 A Sample of the Primitive List.

the operands and attributes corresponding to parameters. It is converted by the Optimizer module, in the CSDE, to the internal format required by succeeding modules [Ref. 2].

In the initial implementation of CSDE there was no front end, i.e., the modules for the designer input and translation to intermediate form did not exist. Therefore, the information required in each primitive was hand generated and formatted by column number so it could be inputed in batch form as a card image file. Because the information is still required in the same format, the output, containing the primitive list and called PRIMFILE, was set up in an identical manner. Each primitive is one line and spaces were added to emulate the blank columns on a punch card. The name of the primitive appears first (software primitives preceded by s., hardware by h.). This is followed by the operand list and selection list (if any) separated by a colon [Ref. 16].

The primitive list, also, contains the design criteria, preceded by d., which allows the designer to specify the order of consideration of the realization volumes. The generation of many realizations can also be requested, each of which is presented along with its chip count and power requirements. The final portion of the primitive list is the application timing table, with each line preceded by a beginning A. This table includes the timing constraints and associated requirements. Further details concerning the information present in each column in the sections of the primitive list are contained in Reference 2 and, therefore, will not be discussed here.

The primitive names, such as eq and mult, were chosen to correspond as closely as possible to the operation suggested, but the names in the realization volumes must be identical to the ones emitted by the translator for the CSDE to function. They are easily modified in the semantic portion of the program if required. However, new operations will dictate a modification to the CSDL grammar with additional productions and semantic rules.

2. Symbol Table

The concept of a symbol table, for use by succeding modules in the CSDE, is a result of this thesis. With the format required in the primitive list, it is difficult for the modules to look up, during the mapping, the attributes such as type, precision, value, and technology for controller variables and constants. It is thought that if this information were available in an easily read form, it would increase the speed of the realization process and raise the level of system efficiency. The functional mapper could read the symbol table first, and generate memory requirements for the controller prior to addressing the operations.

A symbol table can take as many forms as there are formats for a file. The data structure used to hold the information is also arbitrary. At the time of this writing, no decision has been made as to the ideal implementation. So, the information will be dumped into a file in the same format as the primitive list and can later be changed. This information includes all variables with their initial value and precision, all required memory locations, and all constants. In addition, the input and output ports with the expected signal names, technology, and precision are included for possible use. The code to format the symbol table can either be written in a seprate small routine to be installed as a system module, or could be added to the translator. Whichever the case, a decision in this matter may well come in time for the code to be included in the final version of the translator for this thesis, in which case it will be reflected in the example symbol table in the appendices.

3. Scratch Pad

A scratch pad file developed naturally as the translator was designed and implemented. Used initially for output, it significantly aided the debugging and verifying of the processes. Error routines, mentioned previously, send error diagnostics to this file. Traces of parser execution which were developed out of necessity as part of the debugging process also needed an output medium. Because of this, the scratch pad became a formal part of the translator. This file, TRANSLATE, is a text file, with no particular format, containing information which can be helpful to the user. If an error is detected in the input, the diagnostic, which traces the error, will appear here, with comments as to possible corrective action.

In addition, three toggles have been included which provide, if desired, 3 types of traces of program execution. TRACEPARSE will trace the parsing action, providing the transition and reduction numbers as the parser moves through the input. TRACETOK provides the input tokens, one at a time, as they are read. PRINTTABLE displays the controller symbol names with their attributes. These toggles are activated by including "--#" followed by a toggle name at the head of the inut file. More than one toggle per execution can be utilized, but the ensuing report becomes difficult to comprehend.

IV. TESTING AND VALIDATION

A. THEORY OF TESTING

The theory of software testing is a difficult problem and the subject of extensive research. Preliminary reports from this research indicate varying effectiveness. Dijkstra says that debugging can only show the presence of errors, but never their absence [Ref. 12]. It is commonly agreed that program testing cannot assure program "correctness" except under special circumstances. But the debate over Dijkstra's statement continues because others have developed theorems which challange his reasoning [Ref. 13].

The most important factor in testing is to have a well-understood goal for the testing process. In the case of the input translator, proving correctness was not at issue. The algorithm for the parser is well proven to be a correct one [Ref. 9], so the detection of bugs in the program was the goal. Methods of testing include top-down versus bottom-up, static versus dynamic, white box versus black box, and other less known systematic approaches. In bottom-up testing, the idea is to build the program with proven (bug free) components, while top-down begins with tests of the highest level using stubs to simulate the activity of the lower level modules. Static testing attempts to demonstrate the truth of an allegation, i.e., it roughly corresponds to bench-testing a power-driven device without applying power, and dynamic testing seeks to exercise a program in a controlled and systematic way. The white box testing approach is, knowing any part of the sofware system is present for some specific reason, then relating each piece of a software system to the requirement it fulfills. The black box method

is an extensive testing approach which attempts to demonstrate the presence of function by concentrating on the exterior specifications of the software system [Ref. 14].

Just these brief examples illustrate the fact that software testing is not a well-defined science. The primary reason for this is that the concern for software reliability is relatively new. Only in the past decade has any notable effort been expended in understanding how a program can be proven correct or demonstrated to be reliable.

In selecting possible approaches to establish the reliabilty of the input translator, a combination of methods was determined to be the best. As previously noted, program correctness is not the objective. Also, the individual modules were completely debugged as they were built. So the problem reduces to ensuring the function between the input and output is such that the correct output is realized for each possible input. This is a black box methodology, but, since the program is to be exercised as a whole, the concept of dynamic testing also applies. The following section discusses the results of this testing and validation.

B. TEST RESULTS

Functional testing, a form of dynamic testing, involves the testing of a system over each of the different possible classes of input, the testing of each function implemented by the system, and the generation of test output in each of the possible output classes [Ref. 15]. This is the methodology that was employed in the testing of the input translator.

The CSDL input is divided into 5 parts which are IDENTIFICATION, DESIGN CRITERIA, ENVIRONMENT, PROCEDURES, and CONTINGENCIES. Each of these sections was examined in detail to determine the finite set of permutations in structure and content, as set forth by the CSDL grammar.

These possibilities formed the basis for test cases to be used in exercising the input translator. Due to its complexity, the PROCEDURES section received the most attention, but each part is discussed, in terms of the test results, below.

The IDENTIFICATION section, an example of which is contained in figure 4.1, consists of 3 character strings

```
IDENTIFICATION
  DESIGNER: "Thomas H. Carson"
  DATE: "10 April 1984"
  PROJECT: "Start Malfunction Controller"
```

Figure 4.1 IDENTIFICATION Section of the Input.

which make up a portion of the documentation for the system. The strings are not parsed by the input translator, just simply read and ignored. This section is optional, as are they all, and the parser performs no other action on it.

The DESIGN CRITERIA, an example of which is contained in figure 4.2, allows the the user to specify the metric and number of monitors and volumes to be employed in the mapping process, the next module in the CSDE system. The metric is one of 3 choices, all character strings, while the monitors

```
DESIGN CRITERIA
  METRIC FIRST;
  VOLUMES 1;
  MONITORS 1;
```

Figure 4.2 DESIGN CRITERIA Section of the Input.

and volumes are integer values. The parser reads each of these and checks for correctness in terms of value. It then reformats the information and places it in a special portion

```
3d:FIRST     :          1:          1:
```

Figure 4.3 Primitive List Form of the DESIGN CRITERIA.

of the primitive list. The possible alternatives were exercised and the corresponding correct output was generated, an example of which is contained in figure 4.3.

The ENVIRONMENT section, an example of which is contained in figure 4.4, contains the variable declarations for the input. It has 4 parts which are: procedure declarations, input signals, output signals, and duplex signals. While each of the above is optional, the controller will have to sense at least one input and emit one output to have some function. Each declaration can have its name, structure, precision, initial value, or the technology associated

```
ENVIRONMENT
  INPUT: RPM,8,TTL; FIRE_SENSE,1,TTL;
         OIL_PRES,8,TTL; END INPUT;
  OUTPUT: SD1,1,TTL; SD2,1,TTL;
          FIRE_EXT,1,TTL; END OUTPUT;
  ARITHMETIC: STAGFLG,1; END ARITHMETIC;
```

Figure 4.4 ENVIRONMENT Section of the Input.

with it. The type of declaration decides which and how many of the attributes each variable will have. For the internal

program variables, in the ARITHMETIC section, the translator generates a system variable primitive and for each of the signals it produces the associated software primitive. Again, while the translator is parsing the structure of the input, the real work done is reformating the names and their associated attributes into the appropriate primitive. No errors could be detected in exercising the program over this

```
4s.inputport (RPM,TTL:8)
5s.inputport (FIRE_SENSE,TTL:1)
6s.inputport (OIL_PRES,TTL:8)
7s.outputport(FIRE_EXT,TTL:1)
8s.var       (STAGFLG:1,0)
```

Figure 4.5 Primitive List Form of the Input.

portion of the input. An example of the output generated by the above example is contained in figure 4.5.

The PROCEDURES section contains the functions and tasks which establish the purpose of the controller being realized. The differences between functions and tasks are: functions are allowed only one basic statement and return a value while tasks allow multiple statements and perform a job. The key to both is the basic statement which is one of several types seen in most programming languages. The alternatives are: if-then, while-do, for loop, assignment, data input, data output, perform task, and wait. The only ones that might not be familiar are the "perform task" and "wait". "Perform task" allows for nested procedures and the "wait" statement causes the program to suspend itself for a prescribed period of time. An example of a task is contained in figure 4.6.

```
TASK OVRSPD;
 IF START SWIT THEN
  SENSE (RPM);
  SD5 := 0;
  IF RPM > 74 THEN SD5 := 1; END IF;
  ISSUE (SD5);
 END IF;
END OVRSPD;
```

Figure 4.6 PROCEDURES Section Input Example.

This section is where the parser really does its work. It must parse each statement in the procedure and generate an appropriate section of primitives, including temporaries, assembly-language-like software primitives and labels, which fulfill the intent of the statement. Each of the basic statements was exercised through the translator without any error detection, but because of the increased complexity of this section , 100% reliability cannot be confirmed. To do so would require a technique such as path testing. This requires that every logical path through a program be tested at least once. Another possibility is to construct test data which causes each branch in the program to be traversed [Ref. 15]. Alogorithms for such testing exist, but the problems with each are the excessive time, CPU service, and output verification required. Therefore, complete path or branch testing was not attempted. However, the author's confidence in the correctness, after the testing that was conducted, is 100%. An output produced from the above input example is contained in figure 4.7.

The CONTINGENCY LIST section, an example of which is contained in figure 4.8, sets up the flow and timing for the controller by establishing how often each procedure should be executed. There are 4 types of statements allowed in this section: when-do, at time, simple do, and the every.

```
106t.generated for:OVRSPD
107s.proc       (OVRSPD:)
108s.jmpf        (START_SWIT,@12:1)
109s.sensecond   (RPM:8)
110s.gt          (@T01,RPM,@C07:8,8,8)
111s.jmpf        (@T01,@13:8)
112s.assign      (SD5,@C01:1,8)
113s.loc         (@13:)
114s.issuevent   (SD5:1)
115s.loc         (@12:)
116s.exitproc    (OVRSPD:)
```

Figure 4.7 Primitive List Form of the PROCEDURES Section.

While the parsing action for these statements, basically just a reformating routine, is simple, problems developed in

```
CONTINGENCY LIST
 WHEN RESET_SWIT:100MS DO INIT;
 EVERY 1000MS DO CLOCK;
 EVERY 100MS DO OVRSPD;
```

Figure 4.8 CONTINGENCY LIST Input Example.

determining what the format in the primitive list should be. For consistancy, each was treated the same with blanks left in the columns in the "simple do", "every", and "at time" statements where a contingency name appears in the "when-do" statement. Examination of the example output in figure 4.9 and its comparison with the input example above will clarify this point. Alternative entries, such as the word "each", to replace the blanks are under consideration. The desision, based on the requirements of the functional mapper, will not occur prior to the submission of this thesis. Therefore, it is not possible to establish complete

```
A   1 :RESET_SWIT:INIT      :MS: 100,   0,   0,   0,   0
A   2 :           :CLOCK     :MS:1000,   0,   0,   0,   0
A   3 :           :OVRSPD    :MS: 100,   0,   0,   0,   0
```

Figure 4.9 Primitive List Form of the CONTINGENCY LIST.

correctness in this section. The program does act according to its given requirements but the possiblity exists for these to change and, at that time, the section will have to be reverified.

Only the primitive list has been discussed above as program output. The translator also generates a symbol table and a scratch pad. Since the symbol table is unformatted, testing established only that the required information was present. The scratch pad is not a functional member of the CSDE and, therefore, was not tested. The error-checking routines within the translator were tested in so far as they were used in creating correct input file for the testing described above. This was considered adequate due to the impending completion of a syntax-directed editor for composing the CSDL input for the translator.

The input translator was built bottom-up in modular form. This methodology and the use of the automatically generated driver for the parsing routines were significant reasons for the relatively error-free results obtained during the testing of the translator as a whole.

V. CONCLUSIONS AND RECOMMENDATIONS

A. PROGRAM MAINTENANCE

The use of the automatic parser generator in providing the control tables for the translator allows the maximum degree of flexibility for program maintenance. The resulting main program is modular and space efficient, with anticipated changes, such as new emissions from the semantic routines, easily included in the system.

The most obvious portion of the system which will undergo modification is the CSDL grammar. Matelan states there is no capability in CSDL for notational extensibility, nor should there be [Ref. 16]. He points out it is doubtful that the average user can design extensions that will cause less harm than good and the provision of a good macro capability and extendable function/task libraries are preferred. This author disagrees. The advantage of using the parser generator is that a new set of tables can be produced for a modified CSDL with virtually no disturbance to the translator. The changes to the grammar must be consistent with the rules governing LL(1) grammars, but such modifications as the inclusion of new tokens or reserved words is within the capability of advanced program language students. In addition, to keep CSDL static, using the function/task libraries to realize new primitive operations as they become technologically feasible, only results in a less efficient controller. It could lead, in the worst case, to the inability to utilize the latest hardware technology available for controller design. This was most certainly not the intent.

One particular change to CSDL needs immediate attention. Many controllers require an analog input or output signal. Matelan, in his work, makes the assumption that analog information must be converted to a digital signal before it reaches the interface [Ref. 16]. If we have the ability to modify CSDL, this is no longer a complex issue. One possibility is to add a new input/output signal type with the CSDE calling for an A/D-D/A converter when this type is mapped to a library.

B. RECOMMENDATIONS FOR CSDE

The flow of information between modules in CSDE, prior to the impending completion of the designer and translator modules, is consistent. Since the system was implemented in Fortran and resides on one machine, there is no problem. But with the completion of the translator, the subject of this thesis, and the designer module having the same time schedule, the problem of interfaces becomes significant. The translator is written in Pascal and, while the ouput is a simple text file, the information flow will not be as it should. In addition, the designer module is being written in the "C" programming language which will generate, possibly, further interface problems. It is therefore recommended that the CSDE system be incorporated, as quickly as possible, into a data base management system such as Oracle. This was previously discussed in Chapter 3 of this thesis and the design of the data base was the subject of earlier research [Ref. 5]. This research pointed out that this redesign of CSDE should help streamline the operation and eliminate any complex programming schemes that were built out of necessity in earlier work. This concept will eliminate, completely, the interface problems, as the information will be input to and accessible from the database as

it passes from one module to the next. An overall improvement in system documentation, efficiency, and usability should result while, at the same time, allowing each module to maintain its individuality and function.

C. SUMMARY

In completing the design and implementation of the input translator, the objective of this thesis has been accomplished. The translator has been tested and fufills the function required of the module in the Computer System Design Environment [Ref. 2]. The additional features added to the translator include an option to monitor the execution sequence in a variety of tracing modes and extensive error checking. While not considered integral parts of the program, these features were useful in its design and testing. As the CSDE evolves, these features might become superfluous, at which time they could be eliminated. But the information provided by the features should be considered prior to that decision.

The design methodology employed in the production of the translator was not inovative, but it allowed the program to be simple and straightforward. The use of the automatic parser generator was a time saver and proved to be a tool which will allow for the ease of future program maintenance as no other could. In one sense, we could say, while generating a module for a computer aided design system, a computer aided programming tool was central in the development.

The translator will reside in the CSDE on the VAX 11/780 VMS operating system. It is a Pascal program with the source and object code available under the filename "CSDL". The user instructions have been fully covered in the previous sections of this thesis.

APPENDIX A
TRANSLATOR INPUT EXAMPLE

```
IDENTIFICATION

  DESIGNER: "Alan Ross"
  DATE: "12-23-83"
  PROJECT: "Dial Process Control Application"

DESIGN CRITERIA

  METRIC FIRST;
  VOLUMES 8;
  MONITORS 8;

ENVIRONMENT

  INPUT: CONSIN,8,TTL; CONST,8,TTL; FLGA,1,TTL;PINA,8,TTL;
         FLGB,1,TTL; PINB,8,TTL; END INPUT;

  OUTPUT: VA,8,TTL; VB,8,TTL; END OUTPUT;

  ARITHMETIC: KCA,8; KCB,8; CNT_B,8; ITIA,8; ITIB,8; AINT,8;
              TDA,8; TDB,8; BINT,8; VSA,8; VSB,8; BDIFF,8;
              PSA,8; PSB,8; CONPTT,8; EA,8; EB,8; KPIA,8;
              EA1,8; EA2,8; EB1,8; EB2,8; KPIB,8;
              END ARITHMETIC;

PROCEDURES

  FUNCTION DATA_A:
    BINARY,1;
    SENSE (FLGA);
    IF FLGA = 1 THEN DATA_A := 1; END IF;
  END DATA_A;

  FUNCTION DATA_B:
    BINARY,1;
```

```
  SENSE (FLGB);
  IF FLGB = 1 THEN DATA_B := 1; END IF;
END DATA_B;

FUNCTION BCNT:
  BINARY,1;
  IF CNT_B >= 4 THEN BCNT := 1; END IF;
END BCNT;

TASK AFIX;
  ARITHMETIC: ADIFF,8; END ARITHMETIC;
  SENSE (PINA);
  EA := PINA*KCA - PSA;
  ADIFF := (3*EA - 4*EA1 + EA2)*5;
  AINT := AINT + EA/KC;
  VA := VSA + KCA*(EA + ITIA*AINT + TDA*ADIFF);
  ISSUE (VA);
  DATA_A := 0;
  EA2 := EA1;
  EA1 := EA;
END AFIX;

TASK B_CALC;
  SENSE (PINB);
  EB := PINB*KCB - PSB;
  BDIFF := (3*EB - 4*EB1 + EB2)*10;
  BINT := BINT + EB/KCB;
  CNTB := CNTB + 1;
  DATA-B := 0;
END B_CALC;

TASK BFIX;
  CNTB := 0;
  VB := VSB + KCB*(EB + ITIB*BINT + TDB*BDIFF);
  ISSUE (VB);
END BFIX;
```

```
  FUNCTION CONFLG:
    BINARY,1;
    SENSE (CONSIN);
    IF CONSIN > 0 THEN CONFLG := 0; END IF;
  END CONFLG;

  TASK CHGCON;
    SENSE (CONST);
    IF CONPTT = 1 THEN KCA := CONST; END IF;
    IF CONPTT = 2 THEN ITIA := 1/CONST; END IF;
    IF CONPTT = 3 THEN TDA := CONST; END IF;
    IF CONPTT = 4 THEN VSA := CONST; END IF;
    IF CONPTT = 5 THEN PSA := CONST; END IF;
    IF CONPTT = 6 THEN AINT := CONST; END IF;
    IF CONPTT = 7 THEN KCB := CONST; END IF;
    IF CONPTT = 8 THEN ITIB := 1/CONST; END IF;
    IF CONPTT = 9 THEN TDB := CONST; END IF;
    IF CONPTT = 10 THEN VSB := CONST; END IF;
    IF CONPTT = 11 THEN PSB := CONST; END IF;
    IF CONPTT = 12 THEN BINT := CONST; END IF;
  END CHGCON;

CONTINGENCY LIST

    WHEN DATA_A :100MS DO AFIX;

    WHEN DATA_B :50MS DO B_CALC;

    WHEN BCNT :100MS DO BFIX;

    WHEN CONFLG DO CHGCON;
```

APPENDIX B
FORMAL SYNTAX OF CSDL

TERMINALS

1. (
2.)
3. *
4. **
5. *ID*
6. *NUMBER*
7. *STRING*
8. +
9. '
10. -
11. .
12. /
13. /=
14. :
15. :=
16. ;
17. <
18. <=
19. =
20. ==
21. =>
22. >
23. >=
24. AND
25. ARITHMETIC

NONTERMINALS

84. <AOP>
85. <ARITHMETIC BODY>
86. <ARITHMETIC DEC>
87. <ARITHMETIC SPEC>
88. <ASSIGNMENT STMT>
89. <AT TIME>
90. <B1>
91. <B2>
92. <BASIC STMT>
93. <BINARY BODY>
94. <BINARY DEC>
95. <BINARY PRECISION>
96. <BINARY SPEC>
97. <CHARACTER REP LIST>
98. <CHARACTER REP>
99. <CODE DEC LIST>
100. <CODE DEC>
101. <CODE ID>
102. <CODE SPEC>
103. <CODE VAR SPEC>
104. <CONTINGENCY DEF>
105. <CONTINGENCY LIST>
106. <CONTROL SYSTEM DESIGN>
107. <DATA INPUT>
108. <DATA OUTPUT>

26. ASCII6
27. ASCII7
28. AT
29. BCD
30. BINARY
31. CODE
32. CONTINGENCY
33. COST
34. CRITERIA
35. DATE
36. DESIGN
37. DESIGNER
38. DO
39. DUPLEX
40. EBCIDIC
41. ECL
42. END
43. ENVIRONMENT
44. EVERY
45. FIRST
46. FOR
47. FROM
48. FUNCTION
49. H
50. IDENTIFICATION
51. IF
52. ITL
53. IN
54. INPUT
55. ISSUE
56. LIST
57. M
58. METRIC
59. MONITORS
60. MS

109. <DEC GP>
110. <DEC>
111. <DECIMAL PRECISION>
112. <DESIGN CRITERIA>
113. <DUPLEX SPEC>
114. <ENVIRONMENT SECTION>
115. <EPISODE TIMING>
116. <EVERY>
117. <EXPR LIST>
118. <EXPRESSION>
119. <EXP_2>
120. <EXP_3>
121. <EXP_4>
122. <FACTOR>
123. <FOR HEAD>
124. <FOR LOOP>
125. <FORMAL PARAM LIST>
126. <FUNCTION>
127. <FUNCTION_HEAD>
128. <ID LIST>
129. <ID SECTION>
130. <IF HEAD>
131. <IF THEN>
132. <INITIAL VALUE>
133. <INPUT SPEC>
134. <LABELED STMT>
135. <LEFT PART LIST>
136. <LIST BODY>
137. <MAX LOOP COUNT>
138. <METRIC>
139. <MOP>
140. <NAME>
141. <NU>
142. <NUMBER LIST>
143. <OUTPUT SPEC>

61. NOT
62. NS
63. OR
64. OUTPUT
65. POWER
66. PROCEDURES
67. PROJECT
68. S
69. SENSE
70. TASK
71. TERM
72. THEN
73. TO
74. TTL
75. UNTIL
76. US
77. VARIABLES
78. VOLUMES
79. WAIT
80. WHEN
81. WHILE
82. {
83. }

144. <PERFORM TASK>
145. <PERIOD>
146. <PI>
147. <PRIMARY>
148. <PROC DEC GP>
149. <PROC DEC>
150. <PROC GP>
151. <PROC SECTION>
152. <PROC>
153. <QUALIFICATION>
154. <RANK>
155. <RELATION>
156. <RELATIONAL OP>
157. <ROE>
158. <SIMPLE DO>
159. <SIMPLE EXP>
160. <STMT GP>
161. <STMT>
162. <STRUCTURE>
163. <SYSTEM GOAL SYMBOL>
164. <TASK LIST>
165. <TASK>
166. <TASK_HEAD>
167. <TECHNOLOGY>
168. <TERM>
169. <TIME MEASURE>
170. <TIME>
171. <TIMED BLOCK>
172. <TIMED_BLOCK_HEAD>
173. <TRANSMISSION BODY>
174. <TRANSMISSION DEC>
175. <WAIT UNTIL>
176. <WAIT>
177. <WAIT_HEAD>
178. <WHEN DO>

179. <WHILE DO>
180. <WHILE HEAD>
181. <WHILE>
182. <ZOPT PROC DEC GP>

THE PRODUCTIONS

```
1. <SYSTEM GOAL SYMBOL> ::= END <CONTROL SYSTEM DESIGN> END

2. <AOP> ::= 3.          / -

4. <MOP> ::= *
5.          / /

6. <RELATIONAL OP> ::= <
7.                     / <=
8.                     / =
9.                     / >
10.                    / >=
11.                    / /=

12. <PRIMARY> ::= *NUMBER*
13.              / *STRING*
14.              / <NAME>
15.              / ( <EXPRESSION> )

16. <FACTOR> ::= <PRIMARY>
17.             / <FACTOR> ** <PRIMARY>

18. <TERM> ::= <FACTOR>
19.           / <TERM> <MOP> <FACTOR>

20. <SIMPLE EXP> ::= <TERM>
21.                  / <AOP> <TERM>
22.                  / NOT TERM
23.                  / <SIMPLE EXP> <AOP> <TERM>

24. <RELATION> ::= <SIMPLE EXP>
25.             / <SIMPLE EXP> <RELATIONAL OP> <SIMPLE EXP>
```

```
26. <EXP_4> ::= <RELATION>
27.            / <EXP_4> AND <RELATION>

28. <EXP_3> ::= <EXP_4>
29.            / <EXP_3> OR <EXP_4>

30. <EXP_2> ::= <EXP_3>
31.            / <EXP_2> => <EXP_3>

32. <EXPRESSION> ::= <EXP_2>
33.                / <EXPRESSION> == <EXP_2>

34. <EXPR LIST> ::=
35.                / <EXPRESSION>
36.                / <EXPR LIST> , <EXPRESSION>

37. <IF THEN> ::= <IF HEAD> THEN <STMT GP> END IF

38. <IF HEAD> ::= IF <EXPRESSION>

39. <WHILE DO> ::= <WHILE HEAD> DO <STMT GP> END WHILE

40. <WHILE HEAD> ::= <WHILE> <EXPRESION> : <MAX LOOP COUNT>

41. <WHILE> ::= WHILE

42. <FOR LOOP> ::= <FOR HEAD> DO <STMT GP> END FOR

43. <FOR HEAD> ::= FOR *ID* FROM <EXPRESSION> TO
                   <EXPRESSION> : <MAX LOOP COUNT>

44. <PERFORM TASK> ::= *ID*
45.                  / *ID* ( <EXPR LIST> : <ID LIST> )

46. <MAX LOOP COUNT> ::= *NUMBER*

47. <LEFT PART LIST> ::= <NAME> :=
48.                    / <LEFT PART LIST> <NAME> :=

49. <ASSIGNMENT STMT> ::= <LEFT PART LIST> <EXPRESSION>

50. <DATA INPUT> ::= SENSE ( <NAME> )
```

```
51. <DATA OUTPUT> ::= ISSUE ( <NAME> )

52. <TIME MEASURE> ::= H
53.                  / M
54.                  / S
55.                  / MS
56.                  / US
57.                  / NS

58. <PERIOD> ::= *NUMBER* <TIME MEASURE>

59. <TIME> ::= <PERIOD>
60.          / <TIME> <PERIOD>

61. <TIMED BLOCK> ::= <TIMED_BLOCK_HEAD> DO <STMT GP> END IN

62. <TIMED_BLOCK_HEAD> ::= IN <PERIOD>

63. <WAIT> ::= WAIT <PERIOD>
64.          / WAIT <EXPRESSION> : <PERIOD>

65. <WAIT UNTIL> ::= <WAIT_HEAD> <EXPRESSION> : <PERIOD>

66. <WAIT_HEAD> ::= WAIT UNTIL

67. <BASIC STMT> ::= <IF THEN>
68.                / <WHILE DO>
69.                / <FOR LOOP>
70.                / <PERFORM TASK>
71.                / <ASSIGNMENT STMT>
72.                / <DATA INPUT>
73.                / <DATA OUTPUT>
74.                / <TIMED BLOCK>
75.                / <WAIT>
76.                / <WAIT UNTIL>

77. <LABELED STMT> ::= *ID* : <BASIC STMT>

78. <STMT> ::= <BASIC STMT>
```

```
79.           / <LABELED STMT>

80. <STMT GP> ::= <STMT> ;
81.             / <STMT GP> <STMT> ;

82. <PROC DEC> ::= <BINARY SPEC>
83.               / <ARITHMETIC SPEC>
84.               / <CODE SPEC>
85.               / <CODE VAR SPEC>

86. <INPUT SPEC> ::= INPUT : <TRANSMISSION BODY> END INPUT

87. <OUTPUT SPEC> ::= OUTPUT : <TRANSMISSION BODY>
                        END OUTPUT

88. <DEC> ::= <PROC DEC>
89.          / <INPUT SPEC>
90.          / <OUTPUT SPEC>
91.          / <DUPLEX SPEC>

92. <DEC GP> ::= <DEC> ;
93.            / <DEC GP> <DEC> ;

94. <DUPLEX SPEC> ::= DUPLEX <TRANSMISSION BODY> END DUPLEX

95. <BINARY SPEC> ::= BINARY : <BINARY BODY> END BINARY

96. <ARITHMETIC SPEC> ::= ARITHMETIC : <ARITHMETIC BODY>
                            END ARITHMETIC

97. <TRANSMISSION BODY> ::= <TRANSMISSION DEC> ;
98.                       / <TRANSMISSION BODY> <TRANSMISSION DEC> ;

99. <TRANSMISSION DEC> ::= *ID* , <BINARY PRECISION> ,
                             <TECHNOLOGY>

100. <BINARY BODY> ::= <BINARY DEC> ;
101.                 / <BINARY BODY> <BINARY DEC> ;

102. <BINARY DEC> ::= *ID* <STRUCTURE> , <BINARY PRECISION>
                        <INITIAL VALUE>
```

```
103. <ARITHMETIC BODY> ::= <ARITHMETIC DEC> ;
104.                    / <ARITHMETIC BODY> <ARITHMETIC DEC> ;

105. <ARITHMETIC DEC> ::= *ID* <STRUCTURE> ,
              <DECIMAL PRECISION> <INITIAL VALUE>

106. <STRUCTURE> ::=
107.                  / ( <NUMBER LIST> )

108. <NUMBER LIST> ::= *NUMBER*
109.                   / <NUMBER LIST> , *NUMBER*

110. <BINARY PRECISION> ::= *NUMBER*

111. <DECIMAL PRECISION> ::= *NUMBER*

112. <INITIAL VALUE> ::=
113.                     / , *NUMBER*

114. <TECHNOLOGY> ::= TTL
115.                  / ECL
116.                  / ITL

117. <CODE VAR SPEC> ::= CODE VARIABLES : <CODE DEC LIST>
                         END CODE VARIABLES

118. <CODE DEC LIST> ::= <CODE DEC> ;
119.                     / <CODE DEC LIST> <CODE DEC> ;

120. <CODE DEC> ::= *ID* : <CODE ID>

121. <CODE SPEC> ::= CODE : *ID* , <BINARY PRECISION> ;
                    <CHARACTER REP LIST> END CODE

122. <CHARACTER REP LIST> ::= <CHARACTER REP> ;
123.                    / <CHARACTER REP LIST> <CHARACTER REP> ;

124. <CHARACTER REP> ::= *ID* : *NUMBER*

125. <CODE ID> ::= *ID*
126.               / ASCII6
```

```
127.              / ASCII7
128.              / EBCIDIC
129.              / BCD

130. <ID LIST> ::=
131.              / *ID*
132.              / <ID LIST> , *ID*

133. <NAME> ::= *ID*
134.           / *ID* ( <EXPR LIST> )
135.           / *ID* { *NUMBER* : *NUMBER* }

136. <FORMAL PARAM LIST> ::=
137.                          / ( <ID LIST> : <ID LIST> )

138. <PROC> ::= <TASK>
139.           / <FUNCTION>

140. <TASK> ::= <TASK_HEAD> ; <ZOPT PROC DEC GP> <STMT GP>
                END *ID*

141. <ZOPT PROC DEC GP> ::=
142.                        / <PROC DEC GP>

143. <PROC DEC GP> ::= <PROC DEC> ;
144.                  / <PROC DEC GP> <PROC DEC> ;

145. <TASK_HEAD> ::= TASK *ID* <FORMAL PARAM LIST>

146. <FUNCTION> ::= <FUNCTION_HEAD> ; <ZOPT PROC DEC GP>
                    <STMT> END *ID*

147. <FUNCTION_HEAD> ::= FUNCTION *ID* <FORMAL PARAM LIST> :
                BINARY , <BINARY PRECISION> <INITIAL VALUE>
148.                  / FUNCTION *ID* <FORMAL PARAM LIST> :
            ARITHMETIC ,<DECIMAL PRECISION> <INITIAL VALUE>

149. <PROC GP> ::= <PROC> ;
150.              / <PROC GP> <PROC> ;

151. <PROC SECTION> ::=
```

```
152.                    / PROCEDURES <PROC GP>
153. <ROE> ::= <PERIOD>
154. <B1> ::= <PERIOD>
155. <B2> ::= <PERIOD>
156. <RANK> ::= <NU>
157.           / <NU> . <PI>
158. <NU> ::= *NUMBER*
159. <PI> ::= *NUMBER*
160. <QUALIFICATION> ::=
161.                    / IF <EXPRESSION>
162. <EPISODE TIMING> ::=
163.                     / : <ROE>
164.                     / : <ROE> , <B1>
165.                     / : <ROE> , <B1> , <B2>
166.                     / : <ROE> , <B1> , <B2> , <RANK>
167. <WHEN DO> ::= <QUALIFICATION> WHEN <NAME>
                  <EPISODE TIMING> DO <TASK LIST>
168. <SIMPLE DO> ::= <QUALIFICATION> DO <TASK LIST> <RANK>
169. <EVERY> ::= <QUALIFICATION> EVERY <ROE> DO <TASK LIST>
170. <AT TIME> ::= <QUALIFICATION> AT <TIME> DO <TASK LIST>
171. <TASK LIST> ::= <NAME>
172.                / <TASK LIST> THEN <NAME>
173. <CONTINGENCY DEF> ::= <WHEN DO>
174.                      / <SIMPLE DO>
175.                      / <EVERY>
176.                      / <AT TIME>
177. <LIST BODY> ::= <CONTINGENCY DEF> ;
```

```
178.                / <LIST BODY> <CONTINGENCY DEF> ;

179. <CONTINGENCY LIST> ::=
180.                      / CONTINGENCY LIST <LIST BODY>

181. <DESIGN CRITERIA> ::=
182.                   / DESIGN CRITERIA METRIC <METRIC> ;
        VOLUMES <NUMBER LIST> ; MONITORS <NUMBER LIST> ;

183. <METRIC> ::= FIRST
184.             / COST
185.             / POWER

186. <ENVIRONMENT SECTION> ::=
187.                         / ENVIRONMENT <DEC GP>

188. <ID SECTION> ::=
189.                / IDENTIFICATION DESIGNER : *STRING*
         DATE : *STRING* PROJECT : *STRING*

190. <CONTROL SYSTEM DESIGN> ::= <ID SECTION>
     <DESIGN CRITERIA> <ENVIRONMENT SECTION> <PROC SECTION>
     <CONTINGENCY LIST>
```

APPENDIX C

PRIMITIVE LIST

```
P   1t.generated for:   SYSTEM            ********************
P   2s.MAIN      ()
P   3d:FIRST     :8,8,0,0,0,0,0,          0:          0:
P   4s.inputport (CONSIN,TTL:8)
P   5s.inputport (CONST,TTL:8)
P   6s.inputport (FLGA,TTL:1)
P   7s.inputport (PINA,TTL:8)
P   8s.inputport (FLGB,TTL:1)
P   9s.inputport (PINB,TTL:8)
P  10s.outputport(VA,TTL:8)
P  11s.outputport(VB,TTL:8)
P  12s.var       (KCA:8,0)
P  13s.var       (KCB:8,0)
P  14s.var       (CNTB:8,0)
P  15s.var       (ITIA:8,0)
P  16s.var       (ITIB:8,0)
P  17s.var       (AINT:8,0)
P  18s.var       (TDA:8,0)
P  19s.var       (TDB:8,0)
P  20s.var       (BINT:8,0)
P  21s.var       (VSA:8,0)
P  22s.var       (VSB:8,0)
P  23s.var       (BDIFF:8,0)
P  24s.var       (PSA:8,0)
P  25s.var       (PSB:8,0)
P  26s.var       (CONPTT:8,0)
P  27s.var       (EA:8,0)
P  28s.var       (EB:8,0)
P  29s.var       (KPIA:8,0)
P  30s.var       (EA1:8,0)
P  31s.var       (EA2:8,0)
P  32s.var       (EB1:8,0)
P  33s.var       (EB2:8,0)
P  34s.var       (KPIB:8,0)
P  35t.generated for:DATAA                ********************
P  36s.proc      (DATAA:)
P  37s.sensecond (FLGA:1)
P  38s.eq        (@T01,FLGA,@C01:8,1,8)
P  39s.jmpf      (@T01,@01:8)
P  40s.assign    (DATAA,@C01:1,8)
P  41s.loc       (@01:)
P  42s.exitproc  (DATAA:)
P  43t.generated for:DATAB                ********************
P  44s.proc      (DATAB:)
P  45s.sensecond (FLGB:1)
P  46s.eq        (@T01,FLGB,@C01:8,1,8)
P  47s.jmpf      (@T01,@02:8)
P  48s.assign    (DATAB,@C01:1,8)
P  49s.loc       (@02:)
P  50s.exitproc  (DATAB:)
P  51t.generated for:BCNT                 ********************
P  52s.proc      (BCNT:)
P  53s.ge        (@T01,CNTB,@C02:8,8,8)
```

```
P  54s.jmpf       (@T01,@03:8)
P  55s.assign     (BCNT,@C01:1,8)
P  56s.loc        (@03:)
P  57s.exitproc   (BCNT:)
P  58t.generated for:AFIX                  ********************
P  59s.proc       (AFIX:)
P  60s.var        (ADIFF:8,0)
P  61s.sensecond  (PINA:8)
P  62s.mult       (@T01,PINA,KCA:8,8,8)
P  63s.sub        (@T01,@T01,PSA:8,8,8)
P  64s.assign     (EA,@T01:8,8)
P  65s.mult       (@T01,@C03,EA:8,8,8)
P  66s.mult       (@T02,@C02,EA1:8,8,8)
P  67s.sub        (@T01,@T01,@T02:8,8,8)
P  68s.add        (@T01,@T01,EA2:8,8,8)
P  69s.mult       (@T01,@T01,@C04:8,8,8)
P  70s.assign     (ADIFF,@T01:8,8)
P  71s.divide     (@T01,EA,KCA:8,8,8)
P  72s.add        (@T01,AINT,@T01:8,8,8)
P  73s.assign     (AINT,@T01:8,8)
P  74s.mult       (@T01,ITIA,AINT:8,8,8)
P  75s.add        (@T01,EA,@T01:8,8,8)
P  76s.mult       (@T02,TDA,ADIFF:8,8,8)
P  77s.add        (@T01,@T01,@T02:8,8,8)
P  78s.mult       (@T01,KCA,@T01:8,8,8)
P  79s.add        (@T01,VSA,@T01:8,8,8)
P  80s.assign     (VA,@T01:8,8)
P  81s.issuevent  (VA:8)
P  82s.assign     (DATAA,@C05:1,8)
P  83s.assign     (EA2,EA1:8,8)
P  84s.assign     (EA1,EA:8,8)
P  85s.exitproc   (AFIX:)
P  86t.generated for:BCALC                 ********************
P  87s.proc       (BCALC:)
P  88s.sensecond  (PINB:8)
P  89s.mult       (@T01,PINB,KCB:8,8,8)
P  90s.sub        (@T01,@T01,PSB:8,8,8)
P  91s.assign     (EB,@T01:8,8)
P  92s.mult       (@T01,@C03,EB:8,8,8)
P  93s.mult       (@T02,@C02,EB1:8,8,8)
P  94s.sub        (@T01,@T01,@T02:8,8,8)
P  95s.add        (@T01,@T01,EB2:8,8,8)
P  96s.mult       (@T01,@T01,@C06:8,8,8)
P  97s.assign     (BDIFF,@T01:8,8)
P  98s.divide     (@T01,EB,KCB:8,8,8)
P  99s.add        (@T01,BINT,@T01:8,8,8)
P 100s.assign     (BINT,@T01:8,8)
P 101s.add        (@T01,CNTB,@C01:8,8,8)
P 102s.assign     (CNTB,@T01:8,8)
P 103s.assign     (DATAB,@C05:1,8)
P 104s.exitproc   (BCALC:)
P 105t.generated for:BFIX                  ********************
P 106s.proc       (BFIX:)
```

```
P 107s.assign     (CNTB,@C05:8,8)
P 108s.mult       (@T01,ITIB,BINT:8,8,8)
P 109s.add        (@T01,EB,@T01:8,8,8)
P 110s.mult       (@T02,TDB,BDIFF:8,8,8)
P 111s.add        (@T01,@T01,@T02:8,8,8)
P 112s.mult       (@T01,KCB,@T01:8,8,8)
P 113s.add        (@T01,VSB,@T01:8,8,8)
P 114s.assign     (VB,@T01:8,8)
P 115s.issuevent  (VB:8)
P 116s.exitproc   (BFIX:)
P 117t.generated  for:CONFLG                ********************
P 118s.proc       (CONFLG:)
P 119s.sensecond  (CONSIN:8)
P 120s.gt         (@T01,CONSIN,@C05:8,8,8)
P 121s.jmpf       (@T01,@04:8)
P 122s.assign     (CONFLG,@C05:1,8)
P 123s.loc        (@04:)
P 124s.exitproc   (CONFLG:)
P 125t.generated  for:CHGCON                ********************
P 126s.proc       (CHGCON:)
P 127s.sensecond  (CONST:8)
P 128s.eq         (@T01,CONPTT,@C01:8,8,8)
P 129s.jmpf       (@T01,@05:8)
P 130s.assign     (KCA,CONST:8,8)
P 131s.loc        (@05:)
P 132s.eq         (@T01,CONPTT,@C07:8,8,8)
P 133s.jmpf       (@T01,@06:8)
P 134s.divide     (@T01,@C01,CONST:8,8,8)
P 135s.assign     (ITIA,@T01:8,8)
P 136s.loc        (@06:)
P 137s.eq         (@T01,CONPTT,@C03:8,8,8)
P 138s.jmpf       (@T01,@07:8)
P 139s.assign     (TDA,CONST:8,8)
P 140s.loc        (@07:)
P 141s.eq         (@T01,CONPTT,@C02:8,8,8)
P 142s.jmpf       (@T01,@08:8)
P 143s.assign     (VSA,CONST:8,8)
P 144s.loc        (@08:)
P 145s.eq         (@T01,CONPTT,@C04:8,8,8)
P 146s.jmpf       (@T01,@09:8)
P 147s.assign     (PSA,CONST:8,8)
P 148s.loc        (@09:)
P 149s.eq         (@T01,CONPTT,@C08:8,8,8)
P 150s.jmpf       (@T01,@10:8)
P 151s.assign     (AINT,CONST:8,8)
P 152s.loc        (@10:)
P 153s.eq         (@T01,CONPTT,@C09:8,8,8)
P 154s.jmpf       (@T01,@11:8)
P 155s.assign     (KCB,CONST:8,8)
P 156s.loc        (@11:)
P 157s.eq         (@T01,CONPTT,@C10:8,8,8)
P 158s.jmpf       (@T01,@12:8)
P 159s.divide     (@T01,@C01,CONST:8,8,8)
```

```
P 160s.assign     (ITIB,@T01:8,8)
P 161s.loc        (@12:)
P 162s.eq         (@T01,CONPTT,@C11:8,8,8)
P 163s.jmpf       (@T01,@13:8)
P 164s.assign     (TDB,CONST:8,8)
P 165s.loc        (@13:)
P 166s.eq         (@T01,CONPTT,@C06:8,8,8)
P 167s.jmpf       (@T01,@14:8)
P 168s.assign     (VSB,CONST:8,8)
P 169s.loc        (@14:)
P 170s.eq         (@T01,CONPTT,@C12:8,8,8)
P 171s.jmpf       (@T01,@15:8)
P 172s.assign     (PSB,CONST:8,8)
P 173s.loc        (@15:)
P 174s.eq         (@T01,CONPTT,@C13:8,8,8)
P 175s.jmpf       (@T01,@16:8)
P 176s.assign     (BINT,CONST:8,8)
P 177s.loc        (@16:)
P 178s.exitproc   (CHGCON:)
A   1 :DATAA      :AFIX      :MS: 100,   0,   0,   0,   0
A   2 :DATAB      :BCALC     :MS:  50,   0,   0,   0,   0
A   3 :BCNT       :BFIX      :MS: 100,   0,   0,   0,   0
A   4 :CONFLG     :CHGCON    :MS:   0,   0,   0,   0,   0
P 179t.generated for:  SYSTEM              **********
P 180s.cons       (@C01:1,8)
P 181s.cons       (@C02:4,8)
P 182s.cons       (@C03:3,8)
P 183s.cons       (@C04:5,8)
P 184s.cons       (@C05:0,8)
P 185s.cons       (@C06:10,8)
P 186s.cons       (@C07:2,8)
P 187s.cons       (@C08:6,8)
P 188s.cons       (@C09:7,8)
P 189s.cons       (@C10:8,8)
P 190s.cons       (@C11:9,8)
P 191s.cons       (@C12:11,8)
P 192s.cons       (@C13:12,8)
P 193s.var        (@T01:8)
P 194s.var        (@T02:8)
```

```
0001        PROGRAM CSDL (DAT,INPUT,OUTPUT,PRIMFILE,TRANSLATE,SYMFILE);
0002         (*This program uses the output from the CSDE syntax-directed*)
0003         (*editor. The ouput is in standard CSDL form and is         *)
0004         (*translated into a primitive list, contained in PRIMFILE,  *)
0005         (*and the results of the parse in the OUTPUT file.          *)
0006
0007        LABEL 99;
0008
0009        CONST
0010         (* Constants generated by the translator as a result of the     *)
0011         (* CSDL syntax being fed through the automatic parser generator.*)
0012
0013          FREDSIZE = 397;  (*ARRAY LENGTH LIMITS*)
0014          NSETSIZE = 205;
0015          LSETSIZE = 63;
0016
0017          LSSIZE = 573;
0018          PRODSIZE = 205;
0019          FTRNSIZE = 397;
0020          TRANSIZE = 1161;
0021          ENTSIZE =396;
0022          LHSSIZE = 190;
0023          LENSIZE = 190;
0024          FSTATE = 7;
0025          VOCSIZE = 182;
0026          FIRSTRESWD = 24;
0027          LASTRESWD = 81;
0028          NUMTERMINALS = 83;
0029
0030          MAXENT =182;  (*MAXIMUM VALUES OF ARRAY ELEMENTS*)
0031          MAXFRED = 206;
0032          MAXFTRN = 1162;
0033          MAXTRAN = 396;
0034          MAXNSET = 62;
0035          MAXPROD = 190;
0036          MAXLS = 81;
0037          MAXLSET = 574;
0038          MAXLEN = 11;
0039          MAXLHS = 182;
0040
0041          (*PROGRAM CONSTANTS*)
0042
0043          ENDTOK =42;       IDTOK = 5;        NUMTOK = 6;
0044          ARROWTOK = 21;    EQUIVALENCE = 20;PWRTOK = 4;
0045          BECOMESTOK = 15; NOTEQTOK = 13;   LESSEQTOK = 18;
0046          GTREQTOK = 23;    STRINGTOK = 7;
0047
0048
0049          MAXSTK = 40; (* SIZE OF STACK USED IN PARSER *)
0050          TABSIZE = 54; (* SIZE OF SYMBOL TABLE *)
0051          LINELENGTH = 120; (* MAX LENGTH OF INPUT LINES *)
0052          PAGESIZE = 53; (* NUMBER OF LINES PRINTED PER PAGE *)
0053          LINERRARRAYSIZE = 10; (* MAX NO OF ERRORS FLAGGED PER INPUT LINE +2 *)
```

```
0054              MAXSTRINGS = 100; (* MAX LENGTH OF STRINGHEAD ARRAY *)
0055              MAXSSTORE = 1000; (* MAX LENGTH OF STRINGSTORE ARRAY *)
0056              TEMPLISTMAX = 25; (* MAX LENGTH OF A UTILITY IN SEMANTIC *)
0057              MAXTEMPS = 20; (* MAX LENGTH OF A UTILITY IN SEMANTIC *)
0058              TPOOL8SIZE = 10; (* CONSTANT USED IN SEMANTIC *)
0059              MAXEVALSTACK = 20;  (* DEPTH OF ARTH EVAL STACK *)
0060              MAXOPS = 10; (* MAX NUMBER OF OPS HELD IN STORAGE FOR PRINTING *)
0061              MAXSELS = 10; (* MAX NUMBER OF SELS HELD IN STORAGE FOR PRINTING *)
0062              CSSMAX = 20; (* MAX LENGTH OF A UTILITY IN SEMANTIC *)
0063
0064
0065            TYPE
0066
0067
0068              (*SYMBOL TABLE TYPES*)
0069
0070              ALFA = VARYING[MAXSSTORE] OF CHAR;
0071              KINDS = (UNDEFINED,RESWD,BINARY,ARITHMETIC,WORD,CHAREP,
0072                       TRANSDEC,TASK,FNCTN);
0073              EXPTYPES = (INT,REEL,BOOL,STNG,ERRORS);
0074              SYMPTR = SYMENTRY;
0075              SYMENTRY = RECORD
0076                           SYMNAME : ALFA;
0077                           LINK : SYMPTR;
0078                           CASE KIND : KINDS OF
0079                  (*1*)    RESWD : (KEY1 : INTEGER);
0080                  (*2*)    BINARY : (PRECISION2, IVAL2 : INTEGER);
0081                  (*3*)    ARITHMETIC : (PRECISION3, IVAL3 : INTEGER);
0082                  (*4*)    WORD : (CODID4 : SYMPTR; STRINGPTR4 : INTEGER);
0083                  (*5*)    CHAREP : (CODID5 : SYMPTR; IVAL5 : INTEGER);
0084                  (*6*)    TRANSDEC : (TYPE6, TECHNOLOGY6 : SYMPTR;
0085                                       PRECISION6 : INTEGER);
0086                  (*7*)    TASK : (PARAMLIST7 : SYMPTR);
0087                  (*8*)    FNCTN : (PARAMLIST8, TYPE8 : SYMPTR;
0088                                    PRECISION8, IVAL8 : INTEGER);
0089                         END;
0090
0091
0092                (*SEMANTIC TYPES*)
0093
0094              SWITCHES = (TRACEPARSE,TRACETOK,PRINTTABLE);
0095              DESCRIPTOR = RECORD    (* DESCRIBES THE CURRENT TOKEN *)
0096                             SYMNAME,TMPNAME : ALFA;
0097                             LINEPOS,
0098                             INTVAL : INTEGER; (* INTEGER VAL OF CURRENT NUMBER *)
0099                             REALVAL : REAL; (* REAL VAL OF CURRENT NUMBER *)
0100                             CHARVAL : CHAR; (* VAL OF CURRENT CHAR LITERAL *)
0101                             SYMLOC : SYMPTR; (* SYMBOL LOC IN SYMBOL TABLE *)
0102                           END;
0103              TAGTYPE = (QNUM,QNAME);
0104
0105
0106            VAR
```

```
0107
0108            PRIMFILE : TEXT;
0109            DAT : TEXT ;
0110            TRANSLATE : TEXT;
0111            SYMFILE : TEXT;
0112
0113
0114
0115
0116
0117            FRED : PACKED ARRAY[1..FREDSIZE] OF 1..MAXFRED;
0118            NSET : PACKED ARRAY[1..NSETSIZE] OF 1..MAXNSET;
0119            LSET : PACKED ARRAY[1..LSETSIZE] OF 1..MAXLSET;
0120            LS   : PACKED ARRAY[1..LSSIZE]   OF 1..MAXLS;
0121            PROD : PACKED ARRAY[1..PRODSIZE] OF 1..MAXPROD;
0122
0123
0124
0125            FTRN : PACKED ARRAY[1..FTRNSIZE] OF 1..MAXFTRN;
0126            TRAN : PACKED ARRAY[1..TRANSIZE] OF 1..MAXTRAN;
0127            ENT  : PACKED ARRAY[1..ENTSIZE]  OF 1..MAXENT;
0128
0129
0130            LEN  : PACKED ARRAY[1..LENSIZE] OF 0..MAXLEN;
0131            LHS  : PACKED ARRAY[1..LHSSIZE] OF 1..MAXLHS;
0132
0133
0134            (* PROCEDURE PARSE *)
0135
0136            NEXTSYM, (* NEXT SYMBOL IN INPUT STREAM *)
0137            NOWSTA, (* CURRENT STATE *)
0138            REDUCTION, (* POSSIBLE REDUCTIONS *)
0139            TRANSITION : INTEGER; (* POSSIBLE TRANSITIONS *)
0140
0141            (* STACK VARIABLES *)
0142
0143            STACK : ARRAY[1..MAXSTK] OF RECORD
0144                STATE : INTEGER; (* STATE STACK *)
0145                TOK : INTEGER; (* TOKEN STACK *)
0146                DES : DESCRIPTOR; (* TOKEN DESCRIPTOR *)
0147                EXPTYPE : EXPTYPES;
0148              END;
0149            STKPTR : INTEGER; (* TOP OF STACK POINTER *)
0150
0151            (* ERROR HANDLING VARIABLES *)
0152
0153            ERRLIST : SET OF 1..58; (* COMPILATION ERROR LIST *)
0154            LINERRORS : ARRAY[1..LINERRARRAYSIZE] OF RECORD
0155                          ERRPOSITION, (* ERROR POSITION *)
0156                          ERRNUM, (* WHICH ERROR *)
0157                          STATE : INTEGER; (* PARSER STATE WHERE ERROR OCCURRED *)
0158                        END;
0159            LINERRPTR,  (* POINTER INTO LINERRORS *)
```

```
0160            MAXLINERRORS : INTEGER;
0161            PROGRAMERRFLAG,
0162            OVERFLOWLOGGED : BOOLEAN;
0163
0164            (* PROCEDURE NEXTSYM *)
0165
0166            SPS : ARRAY[CHAR] OF INTEGER;
0167            CC,                              (* POINTER TO CURRENT CHAR ON LINE *)
0168            LL,                              (* LENGTH OF CURRENT INPUT SYMBOL *)
0169            SOURCELINECOUNT,                 (* INPUT LINE COUNTER *)
0170            PAGENUMBER,
0171            PAGELINECOUNT : INTEGER;
0172            CH : CHAR;                                (* NEXT CHAR IN LINE *)
0173            LINE : ARRAY[1..LINELENGTH] OF CHAR;       (* INPUT LINE BUFFER *)
0174            ZDATE, ZTIME : PACKED ARRAY[1..11] OF CHAR;
0175            LASTTOK : BOOLEAN;
0176            BUFFER : ALFA;
0177
0178             (* SYMBOL TABLE VARIABLES *)
0179
0180            SYMTABLE : ARRAY[1..TABSIZE] OF SYMPTR;
0181            AMULT : REAL;
0182            STRINGHEAD : ARRAY[1..MAXSTRINGS] OF INTEGER;
0183            STRINGSTORE : PACKED ARRAY[1..MAXSSTORE] OF CHAR;
0184
0185              (* SEMANTIC VARIABLES *)
0186
0187            SWITCH : ARRAY[SWITCHES] OF BOOLEAN;
0188            TEMPLIST : ARRAY[1..TEMPLISTMAX] OF SYMPTR;
0189            TLI : INTEGER;
0190            LINECOUNT,
0191            ALINECOUNT : INTEGER;
0192            FIRSTPARAM : SYMPTR;
0193            LABELCOUNT : INTEGER;
0194
0195
0196            TEMPNAME : ARRAY[1..MAXEVALSTACK] OF RECORD
0197                         NAME : ALFA;
0198                         PRECISION : INTEGER;
0199                         INUSE,USED : BOOLEAN;
0200                       END;
0201
0202            EVALSTACK : ARRAY[1..MAXEVALSTACK] OF RECORD
0203                          NAME : ALFA;
0204                          PRECISION : INTEGER;
0205                          END;
0206
0207
0208            ESI : INTEGER;
0209
0210
0211            CONSTANTSTORE : ARRAY[1..CSSMAX] OF RECORD
0212                              VAL : INTEGER;
```

```
0213                        PRECISION : INTEGER;
0214                        NAME : ALFA;
0215                       END;
0216          CSI : INTEGER;
0217
0218
0219          OPSTORE : ARRAY[1..MAXOPS] OF ALFA;
0220          OPI : INTEGER;
0221
0222
0223          SELSTORE : ARRAY[1..MAXSELS] OF INTEGER;
0224          SELI : INTEGER;
0225          NLI : INTEGER;
0226
0227          DESCRIPTION : DESCRIPTOR;
0228
0229
0230          (* INITIALIZED VALUES GENERATED BY THE AUTOMATIC PARSER *)
0231
0232        VALUE
0233
0234          (*LANGUAGE TERMINALS*)
0235          STRINGSTORE := ('(',')','*','*','*','*','I','D','*','*','N','U','M',
0236           'B','E','R','*','*','S','T','R','I','N','G','*','+',',','-','.','/',
0237           '/','=',':',':','=',';','<','<','=','=','=','=','=','>','>','>','=',
0238           'A','N','D','A','R','I','T','H','M','E','T','I','C','A','S','C','I',
0239           'I','6','A','S','C','I','I','7','A','T','B','C','D','B','I','N','A',
0240           'R','Y','C','O','D','E','C','O','N','T','I','N','G','E','N','C','Y',
0241           'C','O','S','T','C','R','I','T','E','R','I','A','D','A','T','E','D',
0242           'E','S','I','G','N','D','E','S','I','G','N','E','R','D','O','D','U',
0243           'P','L','E','X','E','B','C','I','D','I','C','E','C','L','E','N','D',
0244           'E','N','V','I','R','O','N','M','E','N','T','E','V','E','R','Y','F',
0245           'I','R','S','T','F','O','R','F','R','O','M','F','U','N','C','T','I',
0246           'O','N','H','I','D','E','N','T','I','F','I','C','A','T','I','O','N',
0247           'I','F','I','T','L','I','N','I','N','P','U','T','I','S','S','U','E',
0248           'L','I','S','T','M','M','E','T','R','I','C','M','O','N','I','T','O',
0249           'R','S','M','S','N','O','T','N','S','O','R','O','U','T','P','U','T',
0250           'P','O','W','E','R','P','R','O','C','E','D','U','R','E','S','P','R',
0251           'O','J','E','C','T','S','S','E','N','S','E','T','A','S','K','T','E',
0252           'R','M','T','H','E','N','T','O','T','T','L','U','N','T','I','L','U',
0253           'S','V','A','R','I','A','B','L','E','S','V','O','L','U','M','E','S',
0254           'W','A','I','T','W','H','E','N','W','H','I','L','E','(',')',
0255            666 OF ' ');
0256
0257
0258
0259
0260          (*IN STRINGSTORE THE POSITION OF THE BEGINNING OF EACH TERMINAL*)
0261          STRINGHEAD := (1,2,3,4,6,10,18,26,27,28,29,30,31,
0262           33,34,36,37,38,40,41,43,45,46,48,51,61,67,73,75,78,
0263           84,88,99,103,111,115,121,129,131,137,144,147,150,161,166,171,174,
0264           178,186,187,201,203,206,208,213,218,222,223,229,237,239,242,244,246,
0265           252,257,267,274,275,280,284,288,292,294,297,302,304,313,320,324,328,
```

```
0266              333,334,335,16 OF 0);
0267
0268
0269             (* THE EMITS FROM THE CONFIGERATION SETS IN TERMS *)
0270             (* OF TERMINAL NUMBER                            *)
0271             ENT := (3,42,50,106,129,37,42,36,112,14,34,43,114,7,58,
0272              25,30,31,39,54,64,87,96,102,103,109,110,113,133,143,149,66,
0273              151,35,33,45,65,138,14,14,14,77,5,173,174,14,14,110,16,
0274              48,70,126,127,150,152,165,166,32,105,14,16,5,85,86,5,93,
0275              94,5,14,9,42,174,16,173,173,16,5,5,16,152,16,16,56,
0276              7,78,1,162,42,86,16,162,42,94,16,9,5,99,100,6,95,
0277              39,16,42,42,1,125,125,148,149,182,16,182,51,89,104,116,136,
0278              153,158,178,67,6,142,142,9,25,16,9,30,16,95,14,42,100,
0279              16,9,54,64,5,128,14,149,16,5,46,51,53,55,69,79,81,
0280              88,92,107,108,123,124,130,131,134,135,140,144,160,161,171,172,175,
0281              176,177,179,180,181,160,1,5,6,7,8,10,61,84,118,119,120,
0282              121,122,140,147,155,159,168,16,104,28,38,44,80,14,9,16,2,
0283              6,111,95,16,5,26,27,29,40,101,31,16,41,52,74,167,9,
0284              14,25,30,16,1,14,82,5,118,6,145,1,1,6,75,118,145,
0285              38,72,118,140,15,42,161,16,38,118,38,118,42,118,1,71,168,
0286              20,21,63,24,4,13,17,18,19,22,23,84,156,3,12,139,16,
0287              145,170,140,164,145,157,140,7,6,59,9,132,132,5,97,98,77,
0288              5,128,9,9,117,118,5,92,6,47,49,57,60,62,68,76,169,
0289              140,140,14,160,160,15,5,16,160,14,160,14,5,2,117,119,120,
0290              121,155,147,168,159,122,38,145,6,72,141,154,38,14,115,142,6,
0291              14,42,98,16,2,111,95,2,9,14,14,118,2,2,145,42,42,
0292              42,145,42,6,137,164,140,11,164,157,38,16,6,31,16,132,132,
0293              118,128,6,73,46,51,53,81,6,146,9,164,2,83,118,90,145,
0294              14,9,137,91,145,9,154);
0295
0296
0297             (* THE POSITION OF EACH REDUCTION FOR EACH CONFIGURATION SET *)
0298             FRED := (1,1,2,2,2,3,3,4,4,5,5,5,5,6,
0299              6,6,6,6,6,6,6,6,7,8,9,10,11,11,12,13,14,
0300              15,15,16,16,17,18,19,19,19,19,19,19,19,19,19,19,19,
0301              19,20,20,20,21,21,22,22,23,23,23,24,24,24,25,25,25,
0302              26,26,26,26,26,26,26,26,27,27,27,28,29,30,31,31,32,
0303              33,34,34,34,34,34,34,34,35,35,35,35,36,36,36,36,36,
0304              37,37,38,39,39,39,40,40,41,42,42,42,43,43,43,44,44,
0305              45,47,47,48,49,49,50,50,50,50,51,52,52,53,54,54,54,
0306              54,54,55,55,56,57,58,58,58,58,59,61,61,61,61,61,61,
0307              61,62,63,64,65,66,66,67,67,68,69,69,69,70,70,70,71,
0308              71,72,73,73,74,74,74,74,74,75,76,77,78,79,79,79,80,
0309              81,82,83,84,85,86,87,88,89,90,90,90,90,90,90,90,90,
0310              90,91,92,93,94,94,95,96,97,98,99,100,100,101,102,103,104,
0311              105,105,106,106,106,107,108,108,108,108,108,109,109,110,110,110,111,112,
0312              112,113,113,113,113,114,115,116,116,116,116,117,117,117,117,117,117,117,118,
0313              119,120,120,120,120,120,120,121,122,123,124,125,126,126,126,127,128,
0314              128,129,130,130,131,131,132,132,133,134,135,135,135,136,137,137,137,
0315              137,138,139,139,139,139,139,140,142,143,143,143,144,145,146,147,148,
0316              149,150,150,150,150,150,150,151,152,153,153,153,153,153,154,155,155,
0317              156,157,158,159,160,161,162,163,163,164,165,165,166,167,167,167,167,
0318              167,168,168,168,168,169,170,171,172,173,173,174,174,174,175,176,177,
```

```
0319              177,177,177,178,178,179,180,181,182,182,183,184,184,185,186,187,188,
0320              189,190,191,191,191,191,192,193,194,195,196,197,197,198,199,200,200,
0321              201,202,202,202,203,204,205,205,206);
0322
0323
0324             (* THE POSITION OF EACH TRANSITION FOR EACH CONFIGURATION SET *)
0325
0326             FTRN := (1,2,5,6,7,9,10,10,11,13,14,
0327              15,31,33,34,38,39,40,42,45,46,47,47,47,
0328              47,47,62,63,63,63,63,63,71,73,74,74,74,
0329              74,75,78,81,82,83,84,87,88,91,94,95,95,
0330              96,97,97,98,105,106,106,107,108,108,109,110,112,
0331              115,116,118,121,122,123,126,128,129,130,130,133,136,
0332              136,138,140,150,151,151,161,169,170,172,174,175,176,
0333              177,177,178,179,180,180,182,183,186,187,187,188,188,
0334              188,189,190,192,193,193,201,202,232,232,262,280,280,
0335              281,281,288,292,292,292,293,293,295,297,299,299,299,
0336              301,301,301,302,308,309,310,310,314,314,314,314,316,
0337              318,319,319,322,323,341,343,344,345,365,365,365,365,
0338              365,365,366,366,367,367,367,385,386,386,416,417,417,
0339              418,418,418,436,436,437,455,485,503,505,505,505,505,
0340              505,506,514,515,516,517,518,519,519,519,519,529,532,
0341              532,533,536,539,542,544,545,546,547,547,547,549,551,
0342              554,554,554,554,554,554,554,555,555,555,555,555,555,
0343              556,558,559,560,560,579,606,607,608,609,616,616,618,
0344              620,627,627,629,629,659,689,690,691,691,692,693,693,
0345              723,725,755,757,758,760,779,779,782,799,815,830,844,
0346              850,850,850,850,850,850,850,858,871,871,871,878,878,
0347              878,881,881,885,885,886,888,888,888,890,891,891,891,
0348              892,895,896,896,896,898,900,902,905,906,908,908,909,
0349              927,927,927,927,927,927,927,927,928,929,931,961,991,
0350              991,991,991,1021,1023,1053,1055,1055,1055,1057,1058,1059,1060,
0351              1060,1060,1063,1066,1067,1070,1070,1070,1072,1073,1073,1076,1079,
0352              1080,1082,1082,1083,1084,1085,1085,1085,1087,1089,1089,1107,1109,
0353              1110,1112,1112,1112,1112,1113,1114,1115,1115,1116,1116,1116,1117,
0354              1117,1119,1120,1121,1124,1124,1124,1124,1124,1124,1124,1125,1127,
0355              1128,1146,1146,1146,1146,1146,1146,1146,1149,1150,1150,1150,1152,
0356              1153,1153,1155,1158,1158,1159,1159,1162,1162);
0357
0358
0359             (*  TRANSITIONS *)
0360
0361
0362             TRAN := ( 2,3,4,5,6,7,8,9,10,11,12,13,14,15,
0363              16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,
0364              33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,16,17,
0365              18,19,20,21,22,23,24,25,48,28,29,30,31,49,50,51,52,
0366              53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,
0367              70,43,71,72,73,43,74,45,43,75,45,76,77,78,79,50,51,
0368              52,53,80,56,57,81,82,83,84,85,86,87,62,88,89,90,86,
0369              91,65,92,93,94,95,96,97,98,99,100,101,102,43,103,72,43,
0370              104,72,105,106,105,107,16,17,18,22,23,24,25,108,109,110,111,
0371              16,17,18,22,23,24,25,108,109,112,113,114,115,116,117,118,119,
```

```
0372    120,121,122,123,122,124,125,126,127,128,129,130,99,131,132,96,133,
0373    134,135,136,137,138,139,140,141,16,17,18,22,23,24,25,142,143,
0374    144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,
0375    161,162,163,164,165,166,167,168,169,170,171,172,173,144,145,146,147,
0376    148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,174,
0377    165,166,167,168,169,170,171,172,173,175,176,177,178,179,180,181,182,
0378    183,184,185,186,187,188,189,190,191,192,193,113,114,194,116,118,119,
0379    120,195,196,197,198,199,200,201,202,200,203,204,99,205,206,207,208,
0380    209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,224,225,
0381    226,227,175,176,177,178,179,180,181,182,228,184,185,186,187,188,189,
0382    190,191,192,229,230,231,232,175,176,233,178,179,180,181,234,182,235,
0383    184,185,186,187,188,236,189,190,191,192,237,238,175,176,177,178,179,
0384    180,181,182,239,184,185,186,187,240,189,190,191,192,241,144,242,145,
0385    146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,
0386    163,243,166,167,168,169,170,171,172,173,244,245,175,176,177,178,179,
0387    180,181,182,246,184,185,186,187,188,189,190,191,192,247,175,176,177,
0388    178,179,180,181,182,248,184,185,186,187,188,189,190,191,192,144,249,
0389    145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,
0390    162,163,243,166,167,168,169,170,171,172,173,175,176,177,178,179,180,
0391    181,182,250,184,185,186,187,188,189,190,191,192,251,226,252,175,176,
0392    177,178,187,188,189,253,254,255,256,257,258,179,180,259,260,261,262,
0393    263,264,265,266,267,268,269,270,229,271,272,176,273,274,229,275,276,
0394    176,277,278,279,280,281,282,281,283,284,285,286,287,288,139,289,290,
0395    291,175,176,177,178,179,180,181,182,292,293,184,185,186,187,188,189,
0396    190,191,192,294,145,146,147,148,149,150,151,152,295,154,155,156,157,
0397    158,159,161,162,163,166,167,168,169,170,171,172,173,296,297,254,298,
0398    299,300,301,302,303,304,176,305,176,306,298,299,300,301,302,303,304,
0399    307,254,144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,
0400    159,160,161,162,163,308,165,166,167,168,169,170,171,172,173,144,145,
0401    146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,
0402    163,309,165,166,167,168,169,170,171,172,173,254,310,311,312,144,145,
0403    146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,
0404    163,313,165,166,167,168,169,170,171,172,173,314,254,144,145,146,147,
0405    148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,315,
0406    165,166,167,168,169,170,171,172,173,316,254,317,318,254,175,176,177,
0407    178,179,180,181,182,319,293,184,185,186,187,188,189,190,191,192,267,
0408    268,269,175,176,177,178,179,180,181,182,320,185,186,187,188,189,190,
0409    191,192,175,176,177,178,179,180,181,182,321,186,187,188,189,190,191,
0410    192,175,176,177,178,179,180,181,182,322,187,188,189,190,191,192,175,
0411    176,177,178,179,180,181,182,187,188,189,323,191,192,175,176,177,178,
0412    188,324,175,176,177,178,187,188,189,325,175,176,177,178,179,180,181,
0413    182,187,188,189,326,192,175,176,177,178,327,188,189,229,328,329,330,
0414    331,332,333,334,335,336,122,337,338,339,284,340,341,342,343,219,203,
0415    344,99,345,346,347,348,254,224,226,349,175,176,177,178,179,180,181,
0416    182,350,184,185,186,187,188,189,190,191,192,351,352,229,353,144,354,
0417    145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,
0418    162,163,243,166,167,168,169,170,171,172,173,144,355,145,146,147,148,
0419    149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,243,166,
0420    167,168,169,170,171,172,173,144,356,145,146,147,148,149,150,151,152,
0421    153,154,155,156,157,158,159,160,161,162,163,243,166,167,168,169,170,
0422    171,172,173,229,357,144,358,145,146,147,148,149,150,151,152,153,154,
0423    155,156,157,158,159,160,161,162,163,243,166,167,168,169,170,171,172,
0424    173,359,360,346,347,255,256,257,267,268,269,179,180,265,258,176,273,
```

```
0425              361,176,362,363,176,273,364,229,275,365,366,200,367,368,369,370,281,
0426              371,281,372,175,176,177,178,179,180,181,182,373,184,185,186,187,188,
0427              189,190,191,192,139,374,375,254,376,377,378,379,380,331,381,382,331,
0428              383,176,273,384,254,385,219,386,175,176,177,178,179,180,181,182,387,
0429              184,185,186,187,188,189,190,191,192,229,388,389,331,390,254,391,359,
0430              392,229,393,394,395,330,332,396);
0431
0432
0433
0434
0435            NSET := (59,58,61,57,56,51,51,51,51,57,51,51,51,51,
0436              61,51,51,51,52,51,56,51,61,44,44,35,52,48,51,34,55,
0437              34,53,35,35,43,51,35,42,51,34,55,51,51,53,61,51,51,
0438              28,51,35,51,35,35,51,51,19,33,51,49,2,51,51,51,51,
0439              51,51,51,51,51,51,51,51,9,10,10,1,1,53,3,7,6,
0440              4,10,10,5,5,8,54,44,43,51,51,51,51,51,51,51,51,51,
0441              35,51,51,51,51,29,33,19,62,60,31,2,51,51,30,2,32,
0442              29,8,8,2,2,2,2,2,2,1,1,54,40,37,41,60,59,
0443              28,51,51,51,19,19,49,51,51,36,36,36,36,36,36,36,2,
0444              51,32,51,10,3,7,6,5,10,8,5,4,40,45,50,51,51,
0445              35,47,51,51,9,29,51,51,51,51,60,60,51,37,51,60,58,
0446              51,51,35,51,51,19,51,51,51,51,50,50,51,51,9,60,41,
0447              60,60,41,60);
0448
0449
0450
0451            (* REDUCTIONS *)
0452
0453
0454            PROD := (188,181,1,186,151,83,82,84,85,187,91,89,90,88,
0455              179,184,183,185,92,139,152,138,190,106,106,97,93,136,136,141,149,
0456              141,160,103,100,110,94,98,130,145,142,150,176,175,160,180,174,173,
0457              108,96,104,95,101,118,86,87,131,143,44,133,41,71,78,72,73,
0458              69,67,79,70,74,76,75,68,133,12,13,2,3,161,32,30,28,
0459              18,14,16,26,24,20,177,107,111,112,112,125,126,127,129,128,120,
0460              119,115,116,114,99,130,144,34,38,62,12,66,63,49,14,47,80,
0461              34,22,21,11,6,7,8,9,10,4,5,178,59,171,153,162,189,
0462              109,105,102,117,132,35,133,44,77,52,53,55,57,54,56,58,48,
0463              146,81,140,15,33,31,29,27,17,23,25,19,60,158,156,168,113,
0464              122,137,112,112,134,130,51,50,64,65,46,40,170,172,169,163,182,
0465              124,121,123,148,147,36,42,37,61,39,159,157,167,45,135,164,154,
0466              43,165,155,166);
0467
0468
0469            (* LOOK AHEAD SETS *)
0470            LS := (1,5,6,7,1,5,6,7,8,10,61,2,9,14,16,20,28,38,44,72,
0471              73,80,2,3,8,9,10,12,13,14,16,17,18,19,20,21,22,23,24,28,38,44,
0472              63,72,73,80,2,9,14,16,20,21,24,28,38,44,63,72,73,80,2,9,14,16,
0473              20,21,28,38,44,63,72,73,80,2,9,14,16,20,21,28,38,44,72,73,80,2,
0474              8,9,10,13,14,16,17,18,19,20,21,22,23,24,28,38,44,63,72,73,80,2,
0475              3,4,6,8,9,10,12,13,14,15,16,17,18,19,20,21,22,23,24,28,38,44,
0476              63,72,73,80,2,3,4,8,9,10,12,13,14,16,17,18,19,20,21,22,23,24,
0477              28,38,44,63,72,73,80,2,9,14,16,20,28,38,44,72,80,2,3,8,9,10,
```

```
0478              12,13,14,16,17,18,19,20,21,22,23,24,28,38,44,63,72,80,2,9,14,16,
0479              20,21,24,28,38,44,63,72,80,2,9,14,16,20,21,28,38,44,63,72,80,2,
0480              9,14,16,20,21,28,38,44,72,80,2,8,9,10,13,14,16,17,18,19,20,21,
0481              22,23,24,28,38,44,63,72,80,2,3,4,6,8,9,10,12,13,14,15,16,17,
0482              18,19,20,21,22,23,24,28,38,44,63,72,80,2,3,4,8,9,10,12,13,14,
0483              16,17,18,19,20,21,22,23,24,28,38,44,63,72,80,2,9,14,2,3,8,10,
0484              12,13,14,16,17,18,19,20,21,22,23,24,28,38,44,63,72,80,2,14,16,20,
0485              21,24,28,38,44,63,72,80,2,14,16,20,21,28,38,44,63,72,80,2,14,16,
0486              20,28,38,44,72,80,2,14,16,20,21,28,38,44,72,80,2,8,10,13,14,16,
0487              17,18,19,20,21,22,23,24,28,38,44,63,72,80,2,3,4,8,10,12,13,14,
0488              15,16,17,18,19,20,21,22,23,24,28,38,44,63,72,80,2,3,4,8,10,12,
0489              13,14,16,17,18,19,20,21,22,23,24,28,38,44,63,72,80,2,9,16,2,9,
0490              3,4,8,10,12,13,16,17,18,19,20,21,22,23,24,63,3,4,8,10,12,13,
0491              14,17,18,19,20,21,22,23,24,63,5,42,46,51,53,55,69,79,81,5,25,30,
0492              31,46,51,53,55,69,79,81,5,46,51,53,55,69,79,81,5,42,6,9,16,38,
0493              6,16,72,6,16,38,6,72,6,38,9,38,9,14,9,16,9,11,16,38,11,16,
0494              14,16,14,15,16,38,16,25,30,31,32,39,42,54,64,66,28,38,44,80,28,28,
0495              42,44,51,80,32,42,48,70,32,42,32,42,66,32,42,43,66,32,36,42,43,66,
0496              38,42,72);
0497
0498
0499            (* START LOCATION IN LS OF EACH LOOK AHEAD SET *)
0500
0501            LSET := (1,5,12,23,47,61,74,86,108,135,160,170,193,
0502             206,218,229,250,276,300,303,325,337,348,357,367,387,
0503             411,434,437,439,455,471,480,491,499,501,505,508,511,
0504             513,515,517,519,521,522,525,527,529,530,531,533,534,
0505             543,547,553,557,559,562,566,571,572,573,574);
0506
0507
0508
0509
0510            LEN := (3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,3,1,3,1,3,1,2,2,3,1,3,1,3,1,3,
0511             1,3,1,3,0,1,3,5,2,5,4,1,5,8,1,6,1,2,3,2,4,4,1,1,1,1,1,1,2,1,2,5,2,
0512             2,4,4,2,1,1,1,1,1,1,1,1,1,1,1,3,1,1,2,3,1,1,1,1,5,5,1,1,1,1,2,3,4,5,
0513             5,2,3,5,2,3,5,2,3,5,0,3,1,3,1,1,0,2,1,1,1,7,2,3,3,9,2,3,3,1,1,1,1,
0514             1,0,1,3,1,4,6,0,5,1,1,6,0,1,2,3,3,6,8,8,2,3,0,2,1,1,1,1,3,1,1,0,2,
0515             0,2,4,6,8,6,4,5,5,1,3,1,1,1,1,2,3,0,3,0,11,1,1,1,0,2,0,10,5);
0516
0517
0518
0519            (* LEFT HAND SIDE OF EACH CONFIGURATION SET PRODUCTION RULE
0520               BY PRODUCTION NUMBER *)
0521
0522            LHS := (163,84,84,139,139,156,156,156,156,156,156,147,147,147,147,
0523             122,122,168,168,159,159,159,159,155,155,121,121,120,120,119,119,118,
0524             118,117,117,117,131,130,179,180,181,124,123,144,144,137,135,135,88,
0525             107,108,169,169,169,169,169,169,145,170,170,171,172,176,176,175,177,
0526             92,92,92,92,92,92,92,92,92,92,134,161,161,160,160,149,149,
0527             149,149,133,143,110,110,110,110,109,109,113,96,87,173,173,174,93,
0528             93,94,85,85,86,162,162,142,142,95,111,132,132,167,167,167,103,
0529             99,99,100,102,97,97,98,101,101,101,101,101,101,128,128,128,140,140,
0530             140,125,125,152,152,165,182,182,148,148,166,126,127,127,150,150,151,
```

```
0531              151,157,90,91,154,154,141,146,153,153,115,115,115,115,115,178,158,
0532              116,89,164,164,104,104,104,104,136,136,105,105,112,112,138,138,138,
0533              114,114,129,129,106);
0534
0535
0536             (* PARSER INITIALIZATION *)
0537
0538            FUNCTION HASH (SYM :ALFA) : INTEGER;
0539             (* USED IN ENTER TO CREATE THE INDEX IN THE SYMBOL *)
0540             (* TABLE FOR EACH SYMBOL                          *)
0541
0542              VAR  KEY : INTEGER;
0543             BEGIN
0544              KEY := 17*ORD(SYM[1]) + 15*ORD(SYM[2]) + 13*ORD(SYM[3]) +
0545                      3*ORD(SYM[5]) + ORD(SYM[7]);
0546              HASH := ROUND(KEY*AMULT)MOD TABSIZE + 1
0547             END; (*HASH*)
0548
0549
0550            FUNCTION ENTER (SYM : ALFA) : SYMPTR;
0551             (* ENTERS VALUES IN THE SYMBOL TABLE AND *)
0552             (* RETURNS THE POINTER TO THE SYMBOL     *)
0553
0554              VAR PTR : SYMPTR;
0555                  HASHINDEX : INTEGER;
0556             BEGIN
0557              NEW(PTR);
0558              PTR.SYMNAME := SYM;
0559              PTR.KIND := UNDEFINED;
0560              HASHINDEX := HASH(SYM);
0561              PTR.LINK := SYMTABLE[HASHINDEX];
0562              SYMTABLE[HASHINDEX] := PTR;
0563              ENTER := PTR
0564             END; (*ENTER*)
0565
0566
0567            FUNCTION LOOKUP (SYM : ALFA) : SYMPTR;
0568             (* LOOKS UP SYMBOLS IN THE SYMBOL TABLE *)
0569             (* AND RETURNS THE POINTER TO THE SYMBOL DESIRED *)
0570
0571             VAR PTR, SYMLOC : SYMPTR;
0572                 HASHINDEX : INTEGER;
0573
0574             BEGIN
0575              SYMLOC := NIL;
0576              HASHINDEX := HASH(SYM);
0577              PTR := SYMTABLE[HASHINDEX];
0578              WHILE PTR <> NIL DO
0579               IF SYM = PTR.SYMNAME THEN
0580                BEGIN
0581                 SYMLOC := PTR;
0582                 PTR := NIL
0583                END
```

```
0584              ELSE
0585                PTR := PTR.LINK;
0586              LOOKUP := SYMLOC
0587             END; (*LOOKUP*)
0588
0589
0590           PROCEDURE HEADER;
0591            (* PUTS HEADER IN TRANSLATE FILE *)
0592
0593            BEGIN
0594             PAGENUMBER := PAGENUMBER + 1;
0595             WRITELN (TRANSLATE,'CSDL TRANSLATOR  ',' ':50,'PAGE ',PAGENUMBER : 1);
0596             WRITELN (TRANSLATE,'NAVAL POSTGRADUATE SCHOOL',' ':47);
0597             WRITELN (TRANSLATE,ZDATE : 10, ZTIME : 10);
0598             WRITELN(TRANSLATE);
0599             PAGELINECOUNT := 3
0600            END; (*HEADER*)
0601
0602
0603           FUNCTION CHARVAL (NUMBER : INTEGER) : ALFA;
0604            (* USED TO CREATE LOCATION AND TEMPORARY PRIMITIVE VALUES *)
0605
0606            BEGIN
0607             CASE NUMBER OF
0608               0 : CHARVAL := '00';
0609               1 : CHARVAL := '01';
0610               2 : CHARVAL := '02';
0611               3 : CHARVAL := '03';
0612               4 : CHARVAL := '04';
0613               5 : CHARVAL := '05';
0614               6 : CHARVAL := '06';
0615               7 : CHARVAL := '07';
0616               8 : CHARVAL := '08';
0617               9 : CHARVAL := '09';
0618              10 : CHARVAL := '10';
0619              11 : CHARVAL := '11';
0620              12 : CHARVAL := '12';
0621              13 : CHARVAL := '13';
0622              14 : CHARVAL := '14';
0623              15 : CHARVAL := '15';
0624              16 : CHARVAL := '16';
0625              17 : CHARVAL := '17';
0626              18 : CHARVAL := '18';
0627              19 : CHARVAL := '19';
0628              20 : CHARVAL := '20';
0629             END
0630            END; (*CHARVAL*)
0631
0632
0633           PROCEDURE INITIALIZE;
0634            (* INITIALIZES ALL VARIABLES IN THE PROGRAM AND *)
0635            (* CREATES THE STACKS FOR USE IN THE PARSER     *)
0636
```

```
0637              VAR I,J,LEN,IDXCPY : INTEGER;
0638                  K : SWITCHES;
0639                  PTR : SYMPTR;
0640             BEGIN
0641              REWRITE(PRIMFILE);
0642              OPEN(TRANSLATE,RECORDLENGTH := 600);
0643              REWRITE(TRANSLATE);
0644              OPEN(SYMFILE,RECORDLENGTH := 200);
0645              REWRITE(SYMFILE);
0646              RESET(DAT);
0647              CH := ' ';
0648              CC := 0;
0649              LL := 0;
0650              LASTTOK := FALSE;
0651              SOURCELINECOUNT := 0;
0652              PAGENUMBER := 0;
0653              DATE(ZDATE);
0654              TIME(ZTIME);
0655              HEADER;
0656
0657
0658              SPS['('] :=  1;  SPS[')'] :=  2;  SPS['*'] :=  3;
0659              SPS['+'] :=  8;  SPS[','] :=  9;  SPS['-'] := 10;
0660              SPS['.'] := 11;  SPS['/'] := 12;  SPS[':'] := 14;
0661              SPS[';'] := 16;  SPS['<'] := 17;  SPS['='] := 19;
0662              SPS['>'] := 22;  SPS['{'] := 82;  SPS['}'] := 83;
0663
0664
0665              FOR I := 1 TO TABSIZE DO SYMTABLE[I] :=NIL;
0666              AMULT := 160795.0/262144.0*TABSIZE;
0667
0668
0669              FOR I := FIRSTRESWD TO LASTRESWD DO
0670               BEGIN
0671                LEN := STRINGHEAD[I + 1] - STRINGHEAD[I];
0672                J := 1;
0673                BUFFER := STRINGSTORE[STRINGHEAD[I]];
0674                WHILE J <= LEN -1 DO
0675                 BEGIN
0676                  J := J + 1;
0677                  IF J <= 10 THEN
0678                   BUFFER := BUFFER + STRINGSTORE[STRINGHEAD[I] + (J - 1)];
0679                 END;
0680                WHILE J < 10 DO
0681                 BEGIN
0682                  J := J + 1;
0683                  BUFFER := BUFFER + ' '
0684                 END;
0685                PTR := ENTER (BUFFER);
0686                PTR.KIND := RESWD;
0687                PTR.KEY1 := I
0688               END;
0689               PROGRAMERRFLAG := FALSE;
```

```
0690              ERRLIST := [ ];
0691              LINERRPTR := 0;
0692              OVERFLOWLOGGED := FALSE;
0693              MAXLINERRORS := LINERRARRAYSIZE - 2;
0694              NOWSTA := 1;
0695              NEXTSYM := ENDTOK;
0696              STKPTR := 1;
0697              STACK[1].STATE := 1;
0698              STACK[1].TOK := 0;
0699              DESCRIPTION.SYMNAME := '          ';
0700              DESCRIPTION.INTVAL := 0;
0701              DESCRIPTION.CHARVAL := '@';
0702              DESCRIPTION.REALVAL := 0.0;
0703              DESCRIPTION.SYMLOC := NIL;
0704              BUFFER := '               ';
0705
0706              FOR K := TRACEPARSE TO PRINTTABLE DO
0707               SWITCH[K] := FALSE;
0708              TLI := 0;                          (* START SEMANTIC INITIALIZATION *)
0709              FOR I := 1 TO MAXTEMPS DO
0710               BEGIN
0711                TEMPNAME[I].NAME := '@T      ';
0712                TEMPNAME[I].NAME := TEMPNAME[I].NAME + CHARVAL(I);
0713                IF I IN [1..TPOOL8SIZE] THEN
0714                 TEMPNAME[I].PRECISION := 8
0715                ELSE
0716                 TEMPNAME[I].PRECISION := 16;
0717                TEMPNAME[I].INUSE := FALSE;
0718                TEMPNAME[I].USED := FALSE
0719               END;
0720               ESI := 0;
0721               FOR I := 1 TO CSSMAX DO
0722                BEGIN
0723                 CONSTANTSTORE[I].NAME := '@C      ';
0724                 CONSTANTSTORE[I].NAME := CONSTANTSTORE[I].NAME +
0725                                          CHARVAL(I);
0726                 CONSTANTSTORE[I].VAL := 0;         (* ADDED COMMENT *)
0727                 CONSTANTSTORE[I].PRECISION := 8;   (* ADDED COMMENT *)
0728                END;
0729               CSI := 0;
0730               NLI := 0
0731              END; (*INITIALIZE*)
0732
0733            (* PARSING ROUTINES *)
0734
0735           PROCEDURE PUTT (TITLE : ALFA);
0736            (*PLACES TITLE IN PRIMITIVE FILE. *)
0737
0738             BEGIN
0739              LINECOUNT := LINECOUNT + 1;
0740              WRITELN(PRIMFILE,'P',LINECOUNT : 4,'t.generated for:',TITLE : 10,
0741                      ' ' : 10,'********************');
0742             END; (*PUTT*)
```

```
0743
0744
0745            PROCEDURE PUTS (PRIMNAME : ALFA; OPI, SELI : INTEGER);
0746              (*LOADS PRIMITIVE FILE*)
0747
0748              VAR I, J : INTEGER;
0749              BEGIN
0750               LINECOUNT := LINECOUNT + 1;
0751               WRITE (PRIMFILE, 'P', LINECOUNT : 4, 's.', PRIMNAME : 10, '(');
0752               IF OPI > 0 THEN
0753                BEGIN
0754                 FOR I := 1 TO OPI - 1 DO
0755                  BEGIN
0756                   FOR J := 1 TO 10 DO
0757                    IF OPSTORE[I][J] <> ' ' THEN
0758                     WRITE (PRIMFILE, OPSTORE[I][J] : 1);
0759                   WRITE (PRIMFILE, ',');
0760                  END;
0761                 FOR J := 1  TO 10 DO
0762                  IF OPSTORE[OPI][J] <> ' ' THEN
0763                   WRITE (PRIMFILE, OPSTORE[OPI][J] : 1);
0764                 WRITE (PRIMFILE, ':');
0765                END;
0766               IF SELI > 0 THEN
0767                BEGIN
0768                 FOR I := 1 TO SELI - 1 DO
0769                  WRITE (PRIMFILE, SELSTORE[I] : 1, ',');
0770                 WRITE (PRIMFILE, SELSTORE[SELI] : 1, ')');
0771                END
0772               ELSE
0773                WRITE (PRIMFILE, ')');
0774               WRITELN (PRIMFILE)
0775              END; (* PUTS *)
0776
0777
0778            PROCEDURE PUTSYM (PRIMNAME : ALFA; OPI, SELI : INTEGER);
0779              (* LOADS THE SYMBOL TABLE *)
0780
0781              VAR I, J : INTEGER;
0782              BEGIN
0783               WRITE(SYMFILE, 'S.', PRIMNAME : 10, '(');
0784               IF OPI > 0 THEN
0785                BEGIN
0786                 FOR I := 1 TO OPI - 1 DO
0787                  BEGIN
0788                   FOR J := 1 TO 10 DO
0789                    IF OPSTORE[I][J] <> ' ' THEN
0790                     WRITE(SYMFILE, OPSTORE[I][J] : 1);
0791                   WRITE(SYMFILE, ',');
0792                  END;
0793                 FOR J := 1 TO 10 DO
0794                  IF OPSTORE[OPI][J] <> ' ' THEN
0795                   WRITE(SYMFILE, OPSTORE[OPI][J] : 1);
```

```
0796                WRITE(SYMFILE, ':');
0797               END;
0798              IF SELI > 0 THEN
0799               BEGIN
0800                FOR I := 1 TO SELI - 1 DO
0801                 WRITE(SYMFILE, SELSTORE[I] : 1, ',');
0802                WRITE(SYMFILE, SELSTORE[SELI] : 1, ')');
0803               END
0804              ELSE
0805               WRITE(SYMFILE, ')');
0806              WRITELN(SYMFILE)
0807            END;  (* PUTSYM *)
0808
0809          PROCEDURE GETSYM (VAR NEXTSYM : INTEGER; VAR DES : DESCRIPTOR); FORWARD;
0810          PROCEDURE ERROR (ERRORNUM, INTENSITY,PTROFFSET : INTEGER); FORWARD;
0811
0812          PROCEDURE PUTA (CONTINAME, TASKNAME : ALFA; SELI : INTEGER);
0813            (* UNITS (MS),RHO,BETA1,BETA2,ORDER,PI,GAMMA1,GAMMA2,BCKGRD *)
0814            (* LOADS THE TIMING CONSTRAINTS                              *)
0815
0816           VAR I : INTEGER;
0817           BEGIN
0818            ALINECOUNT := ALINECOUNT + 1;
0819            WRITE(PRIMFILE, 'A', ALINECOUNT : 4,' :',CONTINAME : 10, ':',
0820                  TASKNAME : 10,':MS:');
0821            FOR I := 1 TO SELI - 1 DO
0822             WRITE(PRIMFILE, SELSTORE[I] : 4, ',');
0823            WRITELN(PRIMFILE,SELSTORE[SELI] : 4)
0824           END; (* PUTA *)
0825
0826          PROCEDURE PUTD (METRIC : ALFA; NUMVOLUMES,NUMONITORS : INTEGER);
0827            (* LOADS THE DESIGN CRITERIA *)
0828
0829           VAR I : INTEGER;
0830           BEGIN
0831            LINECOUNT := LINECOUNT + 1;
0832            WRITE(PRIMFILE, 'P',LINECOUNT : 4,'d:', METRIC : 10, ':');
0833            FOR I := 1 TO NUMVOLUMES - 1 DO
0834             WRITE(PRIMFILE, SELSTORE[I] : 1, ',');
0835            WRITE(PRIMFILE, SELSTORE[NUMVOLUMES], ':');
0836            FOR I := NUMVOLUMES + 1 TO NUMVOLUMES + NUMONITORS DO
0837             WRITE(PRIMFILE, SELSTORE[I], ',');
0838            WRITELN(PRIMFILE, SELSTORE[NUMVOLUMES + 1 + NUMONITORS], ':')
0839           END; (* PUTD *)
0840
0841          FUNCTION CHECKTYPE (EXP1TYPE, EXP2TYPE : EXPTYPES) : EXPTYPES;
0842           (* CHECKS THE TYPES OF INPUT VALUES *)
0843
0844           BEGIN
0845            IF EXP1TYPE = EXP2TYPE THEN
0846             CHECKTYPE := EXP1TYPE
0847            ELSE
0848             CHECKTYPE := ERRORS
```

```
0849            END; (* CHECKTYPE *)
0850
0851            FUNCTION COMPUTPRE (PRE1, PRE2 : INTEGER) : INTEGER;
0852             (* COMPUTS THE PRECISION REQUIRED FOR THE *)
0853             (* RESULT OF AN ARITHMETIC OPERATION      *)
0854
0855             BEGIN
0856              IF PRE1 > PRE2 THEN
0857               COMPUTPRE := PRE1
0858              ELSE
0859               COMPUTPRE := PRE2
0860             END; (* COMPUTPRE *)
0861
0862            PROCEDURE NEWTEMP (VAR ZPRECISION : INTEGER; VAR TEMP : ALFA);
0863             (* CHECKS TO SEE IF A VARIABLE NAME HAS BEEN PREVIOUSLY *)
0864             (* DEFINED. IF IT HAS, NEWTEMP RETURNS TRUE. IF NOT, IT *)
0865             (* RETURNS FALSE.                                       *)
0866
0867             CONST LAST8 = 10;
0868             VAR I, STOPSEARCH : INTEGER;
0869             BEGIN
0870              CASE ZPRECISION OF
0871               1,2,3,4,5,6,7,8 :
0872                BEGIN
0873                 I := 1;
0874                 STOPSEARCH := LAST8
0875                END;
0876               9,10,11,12,13,14,15,16 :
0877                BEGIN
0878                 I := LAST8 + 1;
0879                 STOPSEARCH := MAXTEMPS
0880                END;
0881               OTHERWISE ERROR(35,1,-1);
0882              END;
0883              WHILE (I <= STOPSEARCH) AND (TEMPNAME[I].INUSE) DO
0884               I := I + 1;
0885              IF I > STOPSEARCH THEN
0886               ERROR(22,1,-1)
0887              ELSE
0888               BEGIN
0889                TEMP := TEMPNAME[I].NAME;
0890                ZPRECISION := TEMPNAME[I].PRECISION;
0891                TEMPNAME[I].INUSE := TRUE;
0892                IF NOT TEMPNAME[I].USED THEN
0893                 TEMPNAME[I].USED := TRUE
0894               END
0895             END; (* NEWTEMP *)
0896
0897            PROCEDURE RETURNTEMP (TEMP : ALFA);
0898             (* RETURNS FALSE IF A TEMP IS NEW *)
0899
0900             VAR I : INTEGER;
0901             BEGIN
```

```
0902              I := 1;
0903              WHILE (TEMPNAME[I].NAME <> TEMP) AND (I <= MAXTEMPS) DO
0904               I := I + 1;
0905              IF I > MAXTEMPS THEN
0906               WRITELN(TRANSLATE, ' FIX THE ERROR IN RETURNTEMP')
0907              ELSE
0908               TEMPNAME[I].INUSE := FALSE
0909             END; (* RETURNTEMP *)
0910
0911
0912
0913            PROCEDURE PRINTEMPS;
0914             (* LOADS THE VARIABLES *)
0915
0916             VAR I : INTEGER;
0917             BEGIN
0918              FOR I := 1 TO MAXTEMPS DO
0919               IF TEMPNAME[I].USED THEN
0920                BEGIN
0921                 OPSTORE[1] := TEMPNAME[I].NAME;
0922                 SELSTORE[1] := TEMPNAME[I].PRECISION;
0923                 PUTS('var        ', 1,1)
0924                END
0925             END; (* PRINTEMPS *)
0926
0927
0928            PROCEDURE PUSHEVALSTACK (ZNAME : ALFA; ZPRECISION : INTEGER);
0929             (* PUSHES A VARIABLE ON TO THE STACK *)
0930
0931             BEGIN
0932              ESI := ESI + 1;
0933              EVALSTACK[ESI].NAME := ZNAME;
0934              EVALSTACK[ESI].PRECISION := ZPRECISION
0935             END; (*PUSHEVALSTACK *)
0936
0937            PROCEDURE POPEVALSTACK (VAR NAME : ALFA; VAR ZPRECISION : INTEGER);
0938             (* POPS A VARIABLE FROM THE STACK *)
0939
0940             BEGIN
0941              NAME := EVALSTACK[ESI].NAME;
0942              ZPRECISION := EVALSTACK[ESI].PRECISION;
0943              IF (NAME[1] = '@') AND (NAME[2] = 'T') THEN
0944               RETURNTEMP(NAME);
0945              ESI := ESI - 1
0946             END; (* POPEVALSTACK *)
0947
0948            PROCEDURE NEWCONS (ZVALUE : INTEGER; VAR ZNAME : ALFA;
0949                               VAR ZPRECISION : INTEGER);
0950             (* CREATES A NEW CONSTANT IF NEEDED. OTHERWISE *)
0951             (* RETURNS THE POINTER TO THE ONE DESIRED      *)
0952
0953             VAR I : INTEGER;
0954             BEGIN
```

```
0955              I := 1;
0956              WHILE (I <= CSI) AND (CONSTANTSTORE[I].VAL <> ZVALUE) DO
0957               I := I + 1;
0958              IF I > CSI THEN
0959               BEGIN
0960                CSI := CSI + 1;
0961                CONSTANTSTORE[CSI].VAL := ZVALUE;
0962                IF (ZVALUE >= -127) AND (ZVALUE <= 128) THEN
0963                 ZPRECISION := 8
0964                ELSE
0965                 IF (ZVALUE >= -32768) AND (ZVALUE <= 32768) THEN
0966                  ZPRECISION := 16
0967                 ELSE
0968                  BEGIN
0969                   ERROR(31,1,-1);
0970                   ZPRECISION := 16
0971                  END;
0972                CONSTANTSTORE[CSI].PRECISION := ZPRECISION;
0973                ZNAME := CONSTANTSTORE[I].NAME
0974               END
0975              ELSE
0976               BEGIN
0977                ZNAME := CONSTANTSTORE[I].NAME;
0978                ZVALUE := CONSTANTSTORE[I].VAL;
0979                ZPRECISION := CONSTANTSTORE[I].PRECISION
0980               END
0981             END; (* NEWCONS *)
0982
0983            PROCEDURE NEWLABEL (VAR LABELNAME : ALFA);
0984
0985             VAR PLACE, LCCOPY : INTEGER;
0986             BEGIN
0987              LABELNAME := '@         ';
0988              LABELCOUNT := LABELCOUNT + 1;
0989              LCCOPY := LABELCOUNT;
0990              IF LCCOPY > 0 THEN
0991               LABELNAME := LABELNAME + CHARVAL(LCCOPY);
0992             END; (*  NEWLABEL *)
0993
0994
0995
0996
0997            FUNCTION FINDTRANSITION (CSTATE,CTOKEN : INTEGER) : INTEGER;
0998              (* CHECKS TO SEE IF ANY TRANSITIONS EXIST *)
0999
1000              VAR I : INTEGER;
1001              BEGIN
1002               FINDTRANSITION := 0;
1003               FOR I := FTRN[CSTATE] TO FTRN[CSTATE + 1] - 1 DO
1004                IF CTOKEN = ENT[TRAN[I]] THEN
1005                 FINDTRANSITION := TRAN[I]
1006              END; (*FINDTRANSITION*)
1007
```

```
1008
1009            FUNCTION FINDREDUCTION (CSTATE,CTOKEN : INTEGER) : INTEGER;
1010              (* CHECKS TO SEE IF ANY REDUCTIONS EXIST *)
1011
1012              VAR I, J : INTEGER;
1013              BEGIN
1014               FINDREDUCTION := 0;
1015               FOR I := FRED[CSTATE] TO FRED[CSTATE + 1] - 1 DO
1016                FOR J := LSET[NSET[I]] TO LSET[NSET[I] + 1] - 1 DO
1017                 IF CTOKEN = LS[J] THEN
1018                  FINDREDUCTION := PROD[I]
1019              END; (*FINDREDUCTION*)
1020
1021            PROCEDURE DOTRANSITION (NEWSTA : INTEGER);
1022               (* GIVEN THE TRANSITION NUMBER, THIS MODULE *)
1023               (* EXECUTES THE TRANSITION AND RETURNS TO   *)
1024               (* PARSE TO CONTINUE THE LOOP               *)
1025
1026              BEGIN
1027               IF SWITCH[TRACEPARSE] THEN
1028                WRITELN(TRANSLATE, ' TRANSITION FROM STATE ', NOWSTA :2,
1029                        ' TO STATE ', NEWSTA :2);
1030               STKPTR := STKPTR + 1;
1031               IF STKPTR <= MAXSTK THEN
1032                BEGIN
1033                 STACK[STKPTR].TOK := NEXTSYM;
1034                 STACK[STKPTR].DES := DESCRIPTION;
1035                 STACK[STKPTR].STATE := NEWSTA;
1036                 NOWSTA := NEWSTA;
1037                 GETSYM(NEXTSYM,DESCRIPTION);
1038                 IF SWITCH[TRACETOK] THEN
1039                  WRITELN(TRANSLATE, ' NEXTSYM = ', NEXTSYM : 2);
1040                END
1041               ELSE
1042                BEGIN
1043                 WRITELN(TRANSLATE,'GOING INTO ERROR IN DOTRAN');
1044                 ERROR(8,5,0)
1045                END
1046              END; (*DOTRANSITION*)
1047
1048            PROCEDURE SEMANTIC (PRODUCTION : INTEGER); FORWARD;
1049            PROCEDURE SEMANTIC1(PRODUCTION : INTEGER); FORWARD;
1050
1051            PROCEDURE DOREDUCTION (PROD : INTEGER);
1052
1053              (* GIVEN THE REDUCTION, THIS MODULE DOES THE REDUCTION *)
1054              (* BY CALLING SEMANTIC WITH THE PARTICULAR CONSTRUCT   *)
1055              (* IT HAS RECOGNIZED. DOREDUCTION THEN REMOVES THE     *)
1056              (* APPROPRIATE NUMBER OF ITEMS FROM THE STACK AND      *)
1057              (* PLACES THE REDUCED VERSION ON THE STACK.            *)
1058
1059              BEGIN
1060               SEMANTIC(PROD);
```

```
1061              STKPTR := STKPTR - LEN[PROD];
1062              IF STKPTR <= MAXSTK THEN
1063               BEGIN
1064                NOWSTA := FINDTRANSITION(STACK[STKPTR].STATE, LHS[PROD]);
1065                IF SWITCH[TRACEPARSE] THEN
1066                 WRITELN(TRANSLATE, ' PRODUCTION ', PROD : 2,
1067                         ' AND TRANSITION FROM STATE ', STACK[STKPTR].STATE : 2,
1068                         ' TO STATE ', NOWSTA : 2);
1069                STKPTR := STKPTR + 1;
1070                STACK[STKPTR].TOK := LHS[PROD];
1071                STACK[STKPTR].STATE := NOWSTA
1072               END
1073              ELSE
1074               BEGIN
1075                WRITELN(TRANSLATE,'GOING INTO ERROR IN DOREDUCTION');
1076                ERROR(9,5,0)
1077               END
1078            END; (* DOREDUCTION *)
1079
1080
1081          PROCEDURE PARSE;
1082
1083            (* REPEATEDLY EXECUTES UNTIL IT TRANSITIONS TO A FINAL       *)
1084            (* STATE. CALLS FINDREDUCTION TO SEE IF ANY REDUCTIONS       *)
1085            (* EXIST. IF ONE DOES, IT DOES IT BY CALLING DOREDUCTION     *)
1086            (* AND GOES BACK TO REPEAT THE LOOP. IF NO REDUCTIONS        *)
1087            (* EXIST, IT CALLS FINDTRANSITION TO SEE IF ANY TRANSITIONS  *)
1088            (* EXIST. IF ONE DOES, IT CALLS DOTRANSITION AND THEN        *)
1089            (* REPEATS THIS LOOP. IF NEITHER EXIST, THEN EITHER AN       *)
1090            (* ERROR EXISTS OR WE HAVE TRANSITIONED TO A FINAL STATE     *)
1091
1092           BEGIN
1093            REPEAT
1094             REDUCTION := FINDREDUCTION(NOWSTA,NEXTSYM);
1095             IF REDUCTION > 0 THEN
1096              DOREDUCTION(REDUCTION)
1097             ELSE
1098              BEGIN
1099               TRANSITION := FINDTRANSITION(NOWSTA,NEXTSYM);
1100               IF TRANSITION <> FSTATE THEN
1101                BEGIN
1102                 IF TRANSITION > 0 THEN
1103                  DOTRANSITION(TRANSITION)
1104                 ELSE
1105                  BEGIN
1106                   WRITELN(TRANSLATE,'GOING INTO ERROR IN PARSE');
1107                   ERROR(5,4,-1)
1108                  END
1109                END
1110               ELSE
1111                NOWSTA := TRANSITION
1112              END
1113            UNTIL NOWSTA =FSTATE
```

```
1114            END; (* PARSE *)
1115
1116
1117
1118
1119          (* ERROR HANDLING ROUTINES *)
1120
1121          PROCEDURE PRINTERRORS;
1122
1123            VAR I : INTEGER;
1124            BEGIN
1125             IF PAGELINECOUNT + 10 > PAGESIZE THEN
1126              WRITELN(TRANSLATE,'1');
1127             I := 1;
1128             WRITELN(TRANSLATE);
1129             WRITELN(TRANSLATE);
1130             WRITELN(TRANSLATE, ' ':25,'PROGRAM ERRORS');
1131             WRITELN(TRANSLATE,' ':25,'**************');
1132             WRITELN(TRANSLATE);
1133             WRITELN(TRANSLATE);
1134             WHILE ERRLIST <> [ ] DO
1135              BEGIN
1136               WHILE NOT (I IN ERRLIST) DO
1137                I := I + 1;
1138               WRITE(TRANSLATE,I : 5);
1139               CASE I OF
1140                1 : WRITELN(TRANSLATE, ' : DIGIT EXPECTED');
1141                2 : WRITELN(TRANSLATE, ' : ERROR IN IDENTIFIER');
1142                3 : WRITELN(TRANSLATE, ' : ERROR IN NUMBER');
1143                4 : WRITELN(TRANSLATE, ' : ERROR IN EXPONENT');
1144                5 : WRITELN(TRANSLATE, ' : IMPROPER CONSTRUCTION, EXPECTED',
1145                            ' SYMBOL LIST FOLLOWS');
1146                6 : WRITELN(TRANSLATE, ' : BASED INTEGERS ARE DEFINED ONLY FOR',
1147                            ' BASES 2 THROUGH 16');
1148                7 : WRITELN(TRANSLATE, ' : CHARACTER NOT DEFINED FOR THIS BASE');
1149                8 : WRITELN(TRANSLATE, ' : STACK OVERFLOW IN DOTRANSITION. ',
1150                            'TO CORRECT, INCREASE CONSTANT MAXSTK.');
1151                9 : WRITELN(TRANSLATE, ' : STACK OVERFLOW IN DOREDUCTION. ',
1152                           'TO CORRECT, INCREASE CONSTANT MAXSTK.');
1153               10: WRITELN(TRANSLATE, ' : END OF FILE ENCOUNTERED.');
1154               11: WRITELN(TRANSLATE, ' : UNKNOWN OR MISPLACED CHARACTER');
1155               12: WRITELN(TRANSLATE, ' : TOO MANY ERRORS ON THIS LINE');
1156               13: WRITELN(TRANSLATE, ' : INTEGER VALUE EXPECTED');
1157               14: WRITELN(TRANSLATE, ' : IDENTIFIER ALREADY DECLARED');
1158               15: WRITELN(TRANSLATE, ' : CHARACTER ALREADY SPECIFIED');
1159               16: WRITELN(TRANSLATE, ' : CHARACTER REPRESENTATION MUST BE AN',
1160                           ' INTEGER VALUE');
1161               17: WRITELN(TRANSLATE, ' : UNKNOWN OR MISSING COMPILER OPTION',
1162                           ' INSTRUCTION');
1163               18: WRITELN(TRANSLATE, ' : REALS NOT IMPLEMENTED');
1164               19: WRITELN(TRANSLATE, ' : OPERATION NOT DEFINED FOR STRING',
1165                           'OPERANDS');
```

```
1166            20: WRITELN(TRANSLATE, ' : TYPE MISMATCH');
1167            21: WRITELN(TRANSLATE, ' : OPERATION NOT DEFINED FOR BOOLEAN',
1168                        ' OPERANDS');
1169            22: WRITELN(TRANSLATE, ' : TOO MANY REQUESTS FOR TEMP NAMES--',
1170                        ' INCREASE CONSTANT MAXTEMPS');
1171            23: WRITELN(TRANSLATE, ' : OPERATION NOT DEFINED FOR NUMERICAL',
1172                        ' OPERANDS');
1173            24: WRITELN(TRANSLATE, ' : PROCEDURE NAME EXPECTED');
1174            25: WRITELN(TRANSLATE, ' : MISMATCHING BEGIN-END PAIR');
1175            26: WRITELN(TRANSLATE, ' : EXPRESSION TYPE MUST BE BOOLEAN');
1176            27: WRITELN(TRANSLATE, ' : ERROR IN FACTOR');
1177            28: WRITELN(TRANSLATE, ' : IDENTIFIER NOT DECLARED');
1178            29: WRITELN(TRANSLATE, ' : INDEX MUST BE DECLARED AS ARITHMETIC');
1179            30: WRITELN(TRANSLATE, ' : THE FOLLOWING ARGUMENT MUST ',
1180                        'BE BINARY');
1181            31: WRITELN(TRANSLATE, ' : CONSTANT TOO LARGE--NOT IN',
1182                        ' -32768..32768');
1183            32: WRITELN(TRANSLATE, ' : IMPROPER ASSIGNMENT');
1184            33: WRITELN(TRANSLATE, ' : PARAMETERS NOT IMPLEMENTED');
1185            34: WRITELN(TRANSLATE, ' : TASK NAME EXPECTED');
1186            35: WRITELN(TRANSLATE, ' : REQUEST FOR A TEMPORARY WITH A',
1187                        ' PRECISION > 16');
1188            36: WRITELN(TRANSLATE, ' : STRUCTURED VARIABLES ARE ',
1189                        'NOT IMPLEMENTED');
1190            37: WRITELN(TRANSLATE, ' : BIT FIELDS ARE NOT IMPLEMENTED');
1191            38: WRITELN(TRANSLATE, ' : EXPRESSION TYPES MUST BE INTEGER');
1192           END;
1193           ERRLIST := ERRLIST - [I]
1194          END
1195        END; (*PRINTERRORS*)
1196
1197
1198      PROCEDURE RECOVER;
1199       (* THIS ATTEMPTS TO RECOVER THE PARSE FROM AN *)
1200       (* ERROR SUCH AS AN UNDEFINED VARIABLE        *)
1201
1202        CONST NUMSOLIDS = 21;
1203        VAR SOLID : ARRAY[1..NUMSOLIDS] OF INTEGER;
1204
1205        FUNCTION SOLIDTOKEN (TOKEN : INTEGER) : BOOLEAN;
1206         VAR LOWERBOUND, UPPERBOUND, INDEX : INTEGER;
1207             FOUND : BOOLEAN;
1208         BEGIN
1209          LOWERBOUND := 1;
1210          UPPERBOUND := NUMSOLIDS;
1211          FOUND := FALSE;
1212          WHILE (LOWERBOUND < UPPERBOUND) AND (NOT FOUND) DO
1213           BEGIN
1214            INDEX := (LOWERBOUND + UPPERBOUND) DIV 2;
1215            IF TOKEN < SOLID[INDEX] THEN
1216             UPPERBOUND := INDEX - 1
1217            ELSE
1218             IF TOKEN > SOLID[INDEX] THEN
```

```
1219                LOWERBOUND := INDEX + 1
1220               ELSE
1221                FOUND := TRUE
1222             END; (*WHILE*)
1223            SOLIDTOKEN := FOUND
1224           END; (* SOLIDTOKEN *)
1225
1226         BEGIN
1227           SOLID[ 1] :=  85;  SOLID[ 2] :=  92;  SOLID[ 3] :=  97;
1228           SOLID[ 4] :=  99;  SOLID[ 5] := 105;  SOLID[ 6] := 109;
1229           SOLID[ 7] := 112;  SOLID[ 8] := 114;  SOLID[ 9] := 117;
1230           SOLID[10] := 128;  SOLID[11] := 129;  SOLID[12] := 135;
1231           SOLID[13] := 136;  SOLID[14] := 142;  SOLID[15] := 148;
1232           SOLID[16] := 150;  SOLID[17] := 151;  SOLID[18] := 160;
1233           SOLID[19] := 164;  SOLID[20] := 170;  SOLID[21] := 173;
1234
1235           WHILE (NOT SOLIDTOKEN(STACK[STKPTR].TOK)) AND
1236                 (STKPTR > 2) DO
1237            STKPTR := STKPTR - 1;
1238           WHILE (FINDTRANSITION(STACK[STKPTR].STATE,NEXTSYM) = 0) AND
1239                 (FINDREDUCTION(STACK[STKPTR].STATE,NEXTSYM) = 0) AND
1240                 (NOT LASTTOK) DO
1241            GETSYM(NEXTSYM,DESCRIPTION);
1242           IF LASTTOK THEN
1243            ERROR(10,5,-1);
1244           NOWSTA := STACK[STKPTR].STATE
1245         END; (*RECOVER *)
1246
1247
1248       PROCEDURE ERROR (*ERRORNUM, INTENSITY, PTROFFSET : INTEGER *);
1249
1250         VAR POSITION : INTEGER;
1251         BEGIN
1252           WRITELN(TRANSLATE,'IN ERROR SLC = ',SOURCELINECOUNT : 3);
1253           IF NOT PROGRAMERRFLAG THEN
1254            PROGRAMERRFLAG := TRUE;
1255           IF LINERRPTR < MAXLINERRORS THEN
1256            BEGIN
1257             LINERRPTR := LINERRPTR + 1;
1258             LINERRORS[LINERRPTR].ERRPOSITION := CC + PTROFFSET;
1259             LINERRORS[LINERRPTR].ERRNUM := ERRORNUM;
1260             LINERRORS[LINERRPTR].STATE := NOWSTA;
1261             ERRLIST := ERRLIST + [ERRORNUM];
1262            END
1263           ELSE
1264            IF NOT OVERFLOWLOGGED THEN
1265             BEGIN
1266              OVERFLOWLOGGED := TRUE;
1267              LINERRPTR := LINERRPTR + 1;
1268              LINERRORS[LINERRPTR].ERRPOSITION := CC + PTROFFSET;
1269              LINERRORS[LINERRPTR].ERRNUM := 12;
1270              ERRLIST := ERRLIST + [12]
1271             END;
```

```
1272              IF INTENSITY = 5 THEN
1273               GOTO 99;
1274              IF INTENSITY = 4 THEN
1275               RECOVER
1276            END; (* ERROR *)
1277
1278
1279          PROCEDURE PRINTLINERRORS;
1280
1281            VAR LINEPOSITION, MARKER : INTEGER;
1282                MORERRORS : BOOLEAN;
1283
1284            PROCEDURE PRINTERROR (X : INTEGER);
1285             VAR I, J, K, SYM : INTEGER;
1286             BEGIN
1287              WITH LINERRORS[X] DO
1288               BEGIN
1289                IF ERRPOSITION = LINEPOSITION + 1 THEN
1290                 BEGIN
1291                  WRITE(TRANSLATE,'',ERRNUM : 2);
1292                  LINEPOSITION := LINEPOSITION + 3
1293                 END
1294                ELSE
1295                 BEGIN
1296                  WRITE(TRANSLATE,' ': (ERRPOSITION - LINEPOSITION) - 1,'',
1297                        ERRNUM : 2);
1298                  LINEPOSITION := ERRPOSITION + 2;
1299                 END;
1300               IF ERRNUM = 5 THEN
1301                BEGIN
1302                 WRITE(TRANSLATE,' ');
1303                 LINEPOSITION := LINEPOSITION + 1;
1304                 FOR I := FTRN[STATE] TO FTRN[STATE + 1] DO
1305                  BEGIN
1306                   SYM := ENT[TRAN[I]];
1307                   IF SYM <= NUMTERMINALS THEN
1308                    BEGIN
1309                     FOR J := STRINGHEAD[SYM] TO STRINGHEAD[SYM + 1] - 1 DO
1310                      BEGIN
1311                       WRITE(TRANSLATE, STRINGSTORE[J] : 1);
1312                       LINEPOSITION := LINEPOSITION + 1
1313                      END;
1314                     WRITE(TRANSLATE,' ');
1315                     LINEPOSITION := LINEPOSITION + 1
1316                    END
1317                  END;
1318                 FOR I := FRED[STATE] TO FRED[STATE + 1] DO
1319                  FOR J := LSET[NSET[I]] TO LSET[NSET[I] + 1] - 1 DO
1320                   BEGIN
1321                    SYM := LS[J];
1322                    FOR K := STRINGHEAD[SYM] TO STRINGHEAD[SYM + 1] DO
1323                     BEGIN
1324                      WRITE(TRANSLATE,STRINGSTORE[K] : 1);
```

```
1325                      LINEPOSITION := LINEPOSITION + 1
1326                     END;
1327                    WRITE(TRANSLATE,' ');
1328                    LINEPOSITION := LINEPOSITION + 1
1329                   END
1330                 END
1331                END
1332               END;
1333
1334              BEGIN
1335               MORERRORS := TRUE;
1336               LINEPOSITION := 0;
1337               MARKER := 1;
1338               WRITE(TRANSLATE,' **** ');
1339               WHILE MORERRORS DO
1340                BEGIN
1341                 WHILE MARKER <= LINERRPTR DO
1342                 BEGIN
1343                 IF LINERRORS[MARKER].ERRPOSITION > LINEPOSITION THEN
1344                  BEGIN
1345                   PRINTERROR(MARKER);
1346                   LINERRORS[MARKER].ERRPOSITION := 0
1347                  END;
1348                 MARKER := MARKER + 1
1349                END;
1350               MARKER := 1;
1351               WHILE (MARKER <= LINERRPTR) AND
1352                     (LINERRORS[MARKER].ERRPOSITION = 0) DO
1353                MARKER := MARKER + 1;
1354               IF MARKER <= LINERRPTR THEN
1355                BEGIN
1356                 WRITELN(TRANSLATE);
1357                 LINEPOSITION := 0;
1358                 WRITE(TRANSLATE,'       ');
1359                 PAGELINECOUNT := PAGELINECOUNT + 1
1360                END
1361               ELSE
1362                BEGIN
1363                 WRITELN(TRANSLATE);
1364                 MORERRORS := FALSE
1365                END
1366               END
1367              END; (* PRINTLINERRORS *)
1368
1369
1370            PROCEDURE GOPRINTTABLE;
1371             (* THIS IS USED BY A TOGGLE IN THE INPUT IF *)
1372             (* THE DETAILS OF THE PARSE ARE DESIRED IN  *)
1373             (* AN OUTPUT FILE CALLED TRANSLATE          *)
1374
1375              VAR PTR : SYMPTR;
1376                  I, J : INTEGER;
1377
```

```
1378              BEGIN
1379               FOR I := 1 TO TABSIZE DO
1380                BEGIN
1381                 PTR := SYMTABLE[I];
1382                 WRITELN(TRANSLATE,I : 5);
1383                 WHILE PTR <> NIL DO
1384                  BEGIN
1385                   WRITE(TRANSLATE,'     ');
1386                   WRITE(TRANSLATE,PTR.SYMNAME : 10, '    ');
1387                   WITH PTR DO
1388                    CASE KIND OF
1389                     UNDEFINED : WRITE(TRANSLATE,'UNDEFINED');
1390                     RESWD : WRITE(TRANSLATE,'RESWD     ', KEY1 : 5);
1391                     BINARY : WRITE(TRANSLATE, 'BINARY     ', PRECISION2 : 5,
1392                                    IVAL2 : 5);
1393                     ARITHMETIC : WRITE(TRANSLATE, 'ARITHMETIC', PRECISION3 : 5,
1394                                        IVAL3 : 5);
1395                     WORD : WRITE(TRANSLATE, 'TEXT      ', CODID4.SYMNAME : 10,
1396                                  STRINGPTR4 : 5);
1397                     CHAREP : WRITE(TRANSLATE, 'CHARREP    ', CODID5.SYMNAME : 10,
1398                                    IVAL5 : 5);
1399                     TRANSDEC : WRITE(TRANSLATE, 'TRANSDEC ', TYPE6.SYMNAME : 10,
1400                                      TECHNOLOGY6.SYMNAME : 10, PRECISION6 : 5);
1401                     TASK : WRITE(TRANSLATE, 'TASK      ');
1402                     FNCTN : WRITE(TRANSLATE, 'FUNCTION    ', TYPE8.SYMNAME : 10,
1403                                   PRECISION8 : 5)
1404                    END;
1405                   WRITELN(TRANSLATE);
1406                   PTR := PTR.LINK
1407                  END;
1408                 WRITELN(TRANSLATE)
1409                END
1410            END; (* GOPRINTTABLE *)
1411
1412          (*--------------*** GET INPUT SYMBOLS ***---------------*)
1413
1414          PROCEDURE GETSYM (*VAR NESTSYM : INTEGER; VAR DES : DESCRIPTOR *);
1415            (* GETS THE NEXT INPUT SYMBOL AND DETERMINES *)
1416            (* THE TYPE AND DESCRIPTION OF THE TOKEN     *)
1417
1418            VAR LEN,                (* LENGTH OF INPUT SYMBOL *)
1419                I,                  (* INDEX *)
1420                BASE,               (* BASE FOR BASED INTEGER *)
1421                NUMDIGITS,          (* NUMBER OF DIGITS IN CURRENT INTEGER *)
1422                SCALE,              (* EXPONENT ADJUSTMENT *)
1423                EXPONENT : INTEGER;(* INTEGER VAL OF EXPONENT *)
1424                SIGN,               (* FLAG INDICATING NEGATIVE EXPONENT *)
1425                DECIMALFLAG,        (* FLAG INDICATING RADIX PT *)
1426                BASEDNUMBER,        (* FLAG INDICATING BASE OTHER THAN 10 *)
1427                ENDSTRING : BOOLEAN; (* FLAG INDICATING END OF STRING *)
1428                FAC, R : REAL;      (* FOR EXPONENT ADJUSTMENT *)
1429
1430            FUNCTION FLIP (FLIPEE : BOOLEAN) : BOOLEAN;
```

```
1431              BEGIN
1432               IF FLIPEE = TRUE THEN
1433                FLIP := FALSE
1434               ELSE
1435                FLIP := TRUE
1436              END; (*FLIP*)
1437
1438             FUNCTION CHARVAL (CH : CHAR) : INTEGER;
1439              BEGIN
1440               IF CH IN ['A'..'F','0'..'9'] THEN
1441                CASE CH OF
1442                 '0' : CHARVAL :=  0;
1443                 '1' : CHARVAL :=  1;
1444                 '2' : CHARVAL :=  2;
1445                 '3' : CHARVAL :=  3;
1446                 '4' : CHARVAL :=  4;
1447                 '5' : CHARVAL :=  5;
1448                 '6' : CHARVAL :=  6;
1449                 '7' : CHARVAL :=  7;
1450                 '8' : CHARVAL :=  8;
1451                 '9' : CHARVAL :=  9;
1452                 'A' : CHARVAL := 10;
1453                 'B' : CHARVAL := 11;
1454                 'C' : CHARVAL := 12;
1455                 'D' : CHARVAL := 13;
1456                 'E' : CHARVAL := 14;
1457                 'F' : CHARVAL := 15
1458                END
1459               ELSE
1460                CHARVAL := 16
1461             END; (* CHARVAL*)
1462
1463             PROCEDURE INCHAR (VAR CH : CHAR);
1464              BEGIN
1465               IF CC = LL THEN
1466                BEGIN
1467                   IF EOF(DAT) THEN
1468                    BEGIN
1469                     WRITELN(TRANSLATE,'GOING INTO ERROR IN INCHAR');
1470                     ERROR(10,5,0)
1471                    END;
1472                   IF LINERRPTR <> 0 THEN
1473                    BEGIN
1474                     PRINTLINERRORS;
1475                     LINERRPTR := 0;
1476                     OVERFLOWLOGGED := FALSE
1477                    END;
1478                   LL := 0;
1479                   CC := 0;
1480                   IF PAGELINECOUNT MOD PAGESIZE = 0 THEN  (* NEW PAGE *)
1481                    BEGIN
1482                     WRITELN(TRANSLATE,'1');
1483                     HEADER
```

```
1484                 END;
1485                SOURCELINECOUNT := SOURCELINECOUNT + 1;
1486                WRITE(TRANSLATE, SOURCELINECOUNT : 5, ' ');
1487                WHILE NOT EOLN(DAT) DO
1488                 BEGIN
1489                  LL := LL + 1;
1490                  READ(DAT,CH);
1491                  WRITE(TRANSLATE,CH);
1492                  LINE[LL] := CH
1493                 END;
1494                LL := LL + 1;
1495                LINE[LL] := ' ';
1496                WRITELN(TRANSLATE);
1497                READLN(DAT);
1498                PAGELINECOUNT := PAGELINECOUNT + 1
1499               END;
1500              CC := CC + 1;
1501              CH := LINE[CC];
1502              IF EOF(DAT) THEN
1503               IF CC = LL THEN
1504                LASTTOK := TRUE
1505            END; (*INCHAR*)
1506
1507          PROCEDURE GETNUMBER (VAR CH : CHAR; BASE : INTEGER;
1508                    DECIMALPART : BOOLEAN; VAR DIGITS, IVALUE : INTEGER;
1509                    VAR RVALUE : REAL);
1510           BEGIN
1511            DIGITS := 1;
1512            IF CHARVAL(CH) >= BASE THEN
1513             ERROR(7,1,0);
1514            IF DECIMALPART THEN
1515             RVALUE := RVALUE * BASE + CHARVAL(CH)
1516            ELSE
1517             IVALUE := CHARVAL(CH);
1518            INCHAR(CH);
1519            WHILE ((BASE = 10) AND (CH IN ['0'..'9',''])) OR
1520                  ((BASE <> 10) AND (CH IN ['0'..'9','A'..'Z',''])) DO
1521             BEGIN
1522              IF CH = '' THEN
1523               INCHAR(CH);
1524              IF DECIMALPART THEN
1525               RVALUE := RVALUE * BASE + CHARVAL(CH)
1526              ELSE
1527               IVALUE := IVALUE * BASE + CHARVAL(CH);
1528              DIGITS := DIGITS + 1;
1529              IF CHARVAL(CH) >= BASE THEN
1530               ERROR(7,1,0);
1531              INCHAR(CH)
1532             END
1533           END; (* GETNUMBER *)
1534
1535
1536          PROCEDURE GETEXPONENT (VAR SIGN : BOOLEAN; VAR EXP : INTEGER);
```

```
1537              VAR NUMDIGITS, INTVAL : INTEGER;
1538                  REALVAL : REAL;
1539              BEGIN
1540               SIGN := FALSE;
1541               INCHAR(CH);
1542               IF CH = '+' THEN
1543                INCHAR(CH)
1544               ELSE
1545                IF CH = '-' THEN
1546                 BEGIN
1547                  SIGN := TRUE;
1548                  INCHAR(CH)
1549                 END;
1550               IF CH IN ['0'..'9'] THEN
1551                BEGIN
1552                 GETNUMBER(CH,10,FALSE,NUMDIGITS,INTVAL,REALVAL);
1553                 EXP := INTVAL
1554                END
1555               ELSE
1556                BEGIN
1557                 ERROR(4,1,0);
1558                 EXP := 0
1559                END
1560              END;
1561
1562             BEGIN (* GETSYM *)
1563              DES.SYMNAME := '          ';
1564              DES.INTVAL := 0;
1565              DES.REALVAL := 0.0;
1566              DES.TMPNAME := '          ';
1567              DES.CHARVAL := '@';
1568              DES.SYMLOC := NIL;
1569              DES.LINEPOS := LINELENGTH;
1570              BUFFER := '          ';
1571              IF EOF(DAT) AND LASTTOK THEN
1572               NEXTSYM := ENDTOK
1573              ELSE
1574               WHILE CH = ' ' DO
1575                INCHAR(CH);
1576              CASE CH OF
1577               'A','B','C','D','E','F','G','H','I',
1578               'J','K','L','M','N','O','P','Q','R',
1579               'S','T','U','V','W','X','Y','Z' :
1580                BEGIN
1581                 I := 1;
1582                 BUFFER[I] := CH;
1583                 INCHAR(CH);
1584                 WHILE CH IN ['A'..'Z','0'..'9',''] DO
1585                  BEGIN
1586                   IF I < 10 THEN
1587                    BEGIN
1588                     I := I + 1;
1589                     BUFFER[I] := CH
```

```
1590                    END;
1591                         IF CH = '' THEN
1592                    BEGIN
1593                     INCHAR(CH);
1594                     IF I < 10 THEN
1595                      BEGIN
1596                       I := I + 1;
1597                       BUFFER[I] := CH
1598                      END;
1599                     IF NOT (CH IN ['A'..'Z','0'..'9']) THEN
1600                      ERROR(2,1,0)
1601                    END;
1602                   INCHAR(CH)
1603                  END;
1604                 WHILE I < 10 DO
1605                  BEGIN
1606                   I := I + 1;
1607                   BUFFER[I] := ' '
1608                  END;
1609                    (* IDENTIFIER OR RESERVED WORD *)
1610                 DES.SYMLOC := LOOKUP(BUFFER);
1611                 IF DES.SYMLOC <> NIL THEN
1612                  BEGIN
1613                   IF DES.SYMLOC.KIND = RESWD THEN
1614                    BEGIN
1615                     NEXTSYM := DES.SYMLOC.KEY1;
1616                     DES.LINEPOS := CC - 1
1617                    END
1618                   ELSE
1619                    NEXTSYM := IDTOK
1620                  END
1621                 ELSE
1622                  BEGIN
1623                   NEXTSYM := IDTOK;
1624                   DES.SYMLOC := ENTER(BUFFER)
1625                  END;
1626                 DES.SYMNAME := DES.SYMLOC.SYMNAME
1627                END; (* CASE OF 'A'..'Z'*)
1628
1629               '0','1','2','3','4','5','6','7','8','9' :
1630               BEGIN
1631                NEXTSYM := NUMTOK;
1632                BASE := 10;
1633                SCALE := 0;
1634                DECIMALFLAG := FALSE;
1635                BASEDNUMBER := FALSE;
1636                GETNUMBER(CH,BASE,DECIMALFLAG,NUMDIGITS,DES.INTVAL,DES.REALVAL);
1637                IF CH = ' ' THEN  (*BASED NUMBER*)
1638                 BEGIN
1639                  BASEDNUMBER := TRUE;
1640                  BASE := DES.INTVAL;
1641                  IF BASE > 16 THEN
1642                   BEGIN
```

```
1643                    ERROR(6,1,-1);
1644                    BASE := 16
1645                   END;
1646                  INCHAR(CH);
1647                  GETNUMBER(CH,BASE,DECIMALFLAG,NUMDIGITS,DES.INTVAL,DES.REALVAL);
1648                  IF CH = '.' THEN
1649                   BEGIN
1650                    DECIMALFLAG := TRUE;
1651                    DES.REALVAL := DES.INTVAL;
1652                    DES.INTVAL := 0;
1653                    DES.CHARVAL := 'R';
1654                    INCHAR(CH);
1655                    GETNUMBER(CH,BASE,DECIMALFLAG,NUMDIGITS,
1656                              DES.INTVAL,DES.REALVAL);
1657                    SCALE := -NUMDIGITS
1658                   END;
1659                  IF CH <> ' ' THEN
1660                   ERROR(3,1,0);
1661                  INCHAR(CH)
1662                 END;
1663                IF CH = '.' THEN (*DECIMAL PART *)
1664                 BEGIN
1665                  INCHAR(CH);
1666                  IF BASEDNUMBER THEN
1667                   ERROR(3,1,-1);
1668                  DES.REALVAL := DES.INTVAL;
1669                  DES.INTVAL := 0;
1670                  DECIMALFLAG := TRUE;
1671                  DES.CHARVAL := 'R';
1672                  GETNUMBER(CH,BASE,DECIMALFLAG,NUMDIGITS,DES.INTVAL,DES.REALVAL);
1673                  SCALE := -NUMDIGITS
1674                 END;
1675                IF CH = 'E' THEN (* EXPONENT PART *)
1676                 BEGIN
1677                  DECIMALFLAG := FALSE;
1678                  DES.CHARVAL := 'R';
1679                  IF SCALE = 0 THEN
1680                   DES.REALVAL := DES.INTVAL;
1681                  GETEXPONENT(SIGN,EXPONENT);
1682                  IF SIGN THEN
1683                   SCALE := SCALE - EXPONENT
1684                  ELSE
1685                   SCALE := SCALE + EXPONENT
1686                 END;
1687                IF SCALE <> 0 THEN  (*ADJUST SCALE *)
1688                 BEGIN
1689                  R := 1;
1690                  SIGN := SCALE < 0;
1691                  SCALE := ABS(SCALE);
1692                  FAC := BASE;
1693                  REPEAT
1694                   IF ODD(SCALE) THEN
1695                    R := R *FAC;
```

```
1696                    FAC := SQR(FAC);
1697                    SCALE := SCALE DIV 2
1698                   UNTIL SCALE = 0;
1699                   IF SIGN THEN
1700                    DES.REALVAL := DES.REALVAL/R
1701                   ELSE
1702                    DES.REALVAL := DES.REALVAL * R
1703                   END
1704                  END; (* CASE OF '0'..'9' *)
1705
1706                 '(',')','+',',',';','.','{','}' :
1707                  BEGIN
1708                   NEXTSYM := SPS[CH];
1709                   DES.LINEPOS := CC;
1710                   INCHAR(CH)
1711                  END;
1712
1713                 '"' : (* STRINGS *)
1714                  BEGIN
1715                   INCHAR(CH);
1716                   ENDSTRING := FALSE;
1717                   WHILE NOT ENDSTRING DO
1718                    BEGIN
1719                     WHILE CH <> '"' DO
1720                      INCHAR(CH);
1721                     INCHAR(CH);
1722                     IF CH = '"' THEN
1723                      INCHAR(CH)
1724                     ELSE
1725                      ENDSTRING := TRUE
1726                    END;
1727                   NEXTSYM := STRINGTOK
1728                  END; (*STRINGS*)
1729
1730                 '-' :
1731                  BEGIN
1732                   INCHAR(CH);
1733                   IF CH = '-' THEN
1734                    BEGIN
1735                     INCHAR(CH);
1736                     IF CH = ' ' THEN
1737                      BEGIN
1738                       INCHAR(CH);
1739                       GETSYM(NEXTSYM,DES);
1740                       IF NEXTSYM <> IDTOK THEN
1741                        ERROR(17,1,-1)
1742                       ELSE
1743                        IF DES.SYMNAME = 'TRACEPARSE' THEN
1744                         SWITCH[TRACEPARSE] := FLIP(SWITCH[TRACEPARSE])
1745                        ELSE IF DES.SYMNAME = 'TRACETOK   ' THEN
1746                         SWITCH[TRACETOK] := FLIP(SWITCH[TRACETOK])
1747                        ELSE IF DES.SYMNAME = 'PRINTTABLE' THEN
1748                         SWITCH[PRINTTABLE] := FLIP(SWITCH[PRINTTABLE])
```

```
1749                        ELSE
1750                         ERROR(17,1,-1)
1751                       END;
1752                      WHILE CC < LL DO
1753                       INCHAR(CH);
1754                      GETSYM(NEXTSYM,DES)
1755                     END
1756                    ELSE
1757                     NEXTSYM := SPS['-']
1758                   END;
1759
1760                  '=' :
1761                   BEGIN
1762                    INCHAR(CH);
1763                    IF CH = '>' THEN
1764                     BEGIN
1765                      NEXTSYM := ARROWTOK;
1766                      INCHAR(CH)
1767                     END
1768                    ELSE
1769                     IF CH = '=' THEN
1770                      BEGIN
1771                       NEXTSYM := EQUIVALENCE;
1772                       INCHAR(CH)
1773                      END
1774                     ELSE
1775                      NEXTSYM := SPS['=']
1776                   END; (*CASE OF '=' *)
1777
1778                  '*' :   .
1779                   BEGIN
1780                    INCHAR(CH);
1781                    IF CH = '*' THEN
1782                     BEGIN
1783                      NEXTSYM := PWRTOK;
1784                      INCHAR(CH)
1785                     END
1786                    ELSE
1787                     NEXTSYM := SPS['*']
1788                   END; (* CASE OF '*' *)
1789
1790                  ':' :
1791                   BEGIN
1792                    INCHAR(CH);
1793                    IF CH = '=' THEN
1794                     BEGIN
1795                      NEXTSYM := BECOMESTOK;
1796                      INCHAR(CH)
1797                     END
1798                    ELSE
1799                     NEXTSYM := SPS[':']
1800                   END; (* CASE OF ':' *)
1801
```

```
1802                '/' :
1803                 BEGIN
1804                  INCHAR(CH);
1805                  IF CH = '=' THEN
1806                   BEGIN
1807                    NEXTSYM := NOTEQTOK;
1808                    INCHAR(CH)
1809                   END
1810                  ELSE
1811                   NEXTSYM := SPS['/']
1812                 END; (* CASE OF '/' *)
1813
1814                '<' :
1815                 BEGIN
1816                  INCHAR(CH);
1817                  IF CH = '=' THEN
1818                   BEGIN
1819                    NEXTSYM := LESSEQTOK;
1820                    INCHAR(CH)
1821                   END
1822                  ELSE
1823                   NEXTSYM := SPS['<']
1824                 END; (* CASE OF '<' *)
1825
1826                '>' :
1827                 BEGIN
1828                  INCHAR(CH);
1829                  IF CH = '=' THEN
1830                   BEGIN
1831                    NEXTSYM := GTREQTOK;
1832                    INCHAR(CH)
1833                   END
1834                  ELSE
1835                   NEXTSYM := SPS['>']
1836                 END; (* CASE OF '>' *)
1837
1838                '''','$',' ','%','','?','@','','' :
1839                 BEGIN
1840                  ERROR(11,1,0);
1841                  INCHAR(CH);
1842                  GETSYM(NEXTSYM,DES)
1843                 END
1844                END  (* CASE *)
1845               END; (* GETSYM *)
1846
1847
1848
1849          PROCEDURE SEMANTIC (*PRODUCTION : INTEGER*);
1850           (* SEMANTIC AND SEMANTIC1 PERFORM THE STACK *)
1851           (* MANIPULATION BASED ON THE CURRENT PRO-   *)
1852           (* DUCTION NUMBER AND WILL EMIT THE PROPER  *)
1853           (* VALUES TO THE PRIMITIVE LIST, SYMBOL     *)
1854           (* TABLE AND TRANSLATE FILES                *)
```

```
1855
1856            VAR PTR : SYMPTR;
1857                TYPVAR : EXPTYPES;
1858                TEMPNAME : ALFA;
1859                PRCSN, I : INTEGER;
1860            BEGIN
1861             CASE PRODUCTION OF
1862              2, 3,                    (* <AOP> ::= +/- *)
1863              4, 5,                    (* <MOP> ::= *// *)
1864              6, 7, 8, 9, 10, 11:      (* <RELATIONAL OP> ::= </<=/=/>/>=//= *)
1865               STACK[STKPTR].DES.INTVAL := PRODUCTION;
1866
1867              12:                      (* <PRIMARY> ::= <NUMBER> *)
1868               IF STACK[STKPTR].DES.CHARVAL = '@' THEN    (*INTEGER*)
1869                BEGIN
1870                 NEWCONS(STACK[STKPTR].DES.INTVAL, TEMPNAME,PRCSN);
1871                 PUSHEVALSTACK(TEMPNAME, PRCSN);
1872                 STACK[STKPTR].EXPTYPE := INT
1873                END
1874               ELSE                                    (* DES.CHARVAL = 'R' *)
1875                BEGIN
1876                 ERROR(18,1,-3);
1877                 PUSHEVALSTACK('JUNK      ',8);
1878                 STACK[STKPTR].EXPTYPE := REEL
1879                END;
1880
1881              13:                     (* <PRIMARY> ::= *STRING* *)
1882               BEGIN
1883                STACK[STKPTR].EXPTYPE := STNG;
1884                PUSHEVALSTACK('STRING    ',8)
1885               END;
1886
1887              14:                     (* <PRIMARY> ::= <NAME> *)
1888               BEGIN
1889                WITH STACK[STKPTR].DES.SYMLOC DO
1890                 CASE KIND OF TRANSDEC,ARITHMETIC,UNDEFINED:
1891                  BEGIN
1892                   IF KIND = TRANSDEC THEN
1893                    PUSHEVALSTACK(SYMNAME,PRECISION6)
1894                   ELSE
1895                    IF KIND = ARITHMETIC THEN
1896                     PUSHEVALSTACK(SYMNAME,PRECISION3)
1897                    ELSE
1898                     BEGIN
1899                      PUSHEVALSTACK(SYMNAME,8);
1900                      ERROR(28,1,-3)
1901                     END;
1902                   STACK[STKPTR].EXPTYPE := INT
1903                  END;
1904                  FNCTN:
1905                   BEGIN
1906                    IF TYPE8.SYMNAME <> 'ARITHMETIC' THEN
1907                     ERROR(27,1,-3);
```

```
1908                PUSHEVALSTACK(SYMNAME,PRECISION8);
1909                STACK[STKPTR].EXPTYPE := INT
1910               END;
1911              OTHERWISE
1912               BEGIN
1913                ERROR(27,1,-3);
1914                PUSHEVALSTACK('JUNK       ',8);
1915                STACK[STKPTR].EXPTYPE := INT
1916               END
1917              END
1918             END;
1919
1920           15:                              (* <PRIMARY> ::= ( <EXPRESSION> ) *)
1921            (* RETAIN INFORMATION ON EXPRESSION *)
1922            BEGIN
1923             STACK[STKPTR - 2] := STACK[STKPTR - 1]
1924            END;
1925
1926           16:                ;  (* <FACTOR> ::= <PRIMARY> *)
1927
1928           17:                ;  (* <FACTOR> ::= <FACTOR>**<PRIMARY> *)
1929
1930           18:                ;  (* <TERM> ::= <FACTOR> *)
1931
1932           19,                        (* <TERM> ::= <TERM> <MOP> <FACTOR> *)
1933           23,              (* <SIMPLE EXP> ::= <SIMPLE EXP> <AOP> <TERM> *)
1934           25:(* <SIMPLE EXP> ::= <SIMPLE EXP> <RELATIONAL OP> <SIMPLE EXP> *)
1935            BEGIN
1936             TYPVAR := CHECKTYPE(STACK[STKPTR - 2].EXPTYPE,
1937                                 STACK[STKPTR].EXPTYPE);
1938             POPEVALSTACK(OPSTORE[3],SELSTORE[3]);
1939             POPEVALSTACK(OPSTORE[2],SELSTORE[2]);
1940             SELSTORE[1] := COMPUTPRE(SELSTORE[2],SELSTORE[3]);
1941             NEWTEMP(SELSTORE[1],OPSTORE[1]);
1942             PUSHEVALSTACK(OPSTORE[1],SELSTORE[1]);
1943             CASE TYPVAR OF
1944              INT: CASE STACK[STKPTR-1].DES.INTVAL OF
1945               2 : (* + *) PUTS('add        ',3,3);
1946               3 : (* - *) PUTS('sub        ',3,3);
1947               4 : (* * *) PUTS('mult       ',3,3);
1948               5 : (* / *) PUTS('divide     ',3,3);
1949               6 : (* < *) PUTS('lt         ',3,3);
1950               7 : (* <= *) PUTS('le         ',3,3);
1951               8 : (* = *) PUTS('eq         ',3,3);
1952               9 : (* > *) PUTS('gt         ',3,3);
1953              10 : (* >= *) PUTS('ge         ',3,3);
1954              11 : (* /= *) PUTS('ne         ',3,3);
1955             END; (*CASE OF STACK*)
1956             STNG : ERROR(19,0,-1);
1957             BOOL : ERROR(21,0,-1);
1958             ERRORS : ERROR(20,0,-1);
1959            END;  (* CASE OF TYPEVAR *)
1960            IF STACK[STKPTR-1].DES.INTVAL IN [6..13] THEN
```

```
1961                STACK[STKPTR-2].EXPTYPE := BOOL;
1962              END;
1963
1964            20 :              (* <SIMPLE EXP> ::= <TERM> *)  ;
1965
1966            21 :              (* <SIMPLE EXP> ::= <AOP><TERM> *)
1967              IF STACK[STKPTR].EXPTYPE <> INT THEN
1968               ERROR(20,1,-1)
1969              ELSE
1970               IF STACK[STKPTR - 1].DES.INTVAL = 3 THEN  (*UNARY MINUS*)
1971                BEGIN
1972                 POPEVALSTACK(OPSTORE[3],SELSTORE[3]);
1973                 NEWCONS(0,OPSTORE[2],SELSTORE[2]);
1974                 SELSTORE[1] := COMPUTPRE(SELSTORE[2],SELSTORE[3]);
1975                 NEWTEMP(SELSTORE[1],OPSTORE[1]);
1976                 PUTS('sub       ',3,3);
1977                 PUSHEVALSTACK(OPSTORE[1],SELSTORE[1])
1978                END;
1979
1980              22 :              (* <SIMPLE EXP> ::= NOT TERM *)
1981                IF STACK[STKPTR].EXPTYPE <> BOOL THEN
1982                  ERROR(20,1,-1)
1983                ELSE
1984                 BEGIN         (*GENERATE NOT*)
1985                  POPEVALSTACK(OPSTORE[2],SELSTORE[2]);
1986                  NEWTEMP(SELSTORE[2], OPSTORE[1]);
1987                        (* PRECISION, TEMPNAME *)
1988                  PUSHEVALSTACK(OPSTORE[1],SELSTORE[1]);
1989                  PUTS('not       ',2,2)
1990                 END;
1991
1992              24 :              (* <RELATION> ::= <SIMPLE EXP> *)  ;
1993
1994              26 :              (* <EXP4> ::= <RELATION> *)        ;
1995
1996              27,               (* <EXP4> ::= <EXP4> AND <RELATION> *)
1997              29,               (* <EXP3> ::= <EXP3> OR <EXP4> *)
1998              31,               (* <EXP2> ::= <EXP-2> => <EXP3> *)
1999              33 :              (* <EXPRESSION> ::= <EXPRESSION> == <EXP2> *)
2000                BEGIN
2001                 TYPVAR := CHECKTYPE(STACK[STKPTR - 2].EXPTYPE,
2002                                     STACK[STKPTR].EXPTYPE);
2003                 POPEVALSTACK(OPSTORE[3],SELSTORE[3]);
2004                 POPEVALSTACK(OPSTORE[2],SELSTORE[2]);
2005                 SELSTORE[1] := COMPUTPRE(SELSTORE[3],SELSTORE[2]);
2006                 NEWTEMP(SELSTORE[1],OPSTORE[1]);
2007                 PUSHEVALSTACK(OPSTORE[1],SELSTORE[1]);
2008                 CASE TYPVAR OF
2009                  BOOL : CASE PRODUCTION OF
2010                    27 : PUTS('and       ',3,3);
2011                    29 : PUTS('or        ',3,3);
2012                    31 : PUTS('implicate ',3,3);
2013                    33 : PUTS('equivalenc',3,3)
```

```
2014                    END;
2015                    INT : ERROR(23,1,-1);
2016                    STNG : ERROR(19,1,-1);
2017                    ERRORS : ERROR(20,1,-1)
2018                  END
2019                 END;
2020
2021                28 :              (* <EXP3> ::= <EXP4> *)    ;
2022
2023                30 :              (* <EXP2> ::= <EXP3> *)    ;
2024
2025                32 :              (* <EXPRESSION> ::= <EXP2> *)  ;
2026
2027                34 :              (* <EXPR LIST> ::=        *)
2028                  STACK[STKPTR + 1].DES.INTVAL := 0;
2029
2030                35 :              (* <EXPR LIST> ::= <EXPRESSION> *)
2031                  STACK[STKPTR].DES.INTVAL := 1;
2032
2033                36 :               (* <EXPR LIST> ::= <EXPR LIST>,<EXPRESSION> *)
2034                  STACK[STKPTR-2].DES.INTVAL := STACK[STKPTR-2].DES.INTVAL + 1;
2035
2036                37 :          (* <IF THEN> ::= <IF HEAD> THEN <STMT GP> END IF *)
2037                  BEGIN
2038                   OPSTORE[1] := STACK[STKPTR - 4].DES.SYMNAME;
2039                   PUTS('loc        ',1,0);
2040                   PUTSYM('LOC        ',1,0)
2041                  END;
2042
2043                38 :             (* <IF HEAD> ::= IF<EXPRESSION> *)
2044                  BEGIN
2045                   IF STACK[STKPTR].EXPTYPE <> BOOL THEN
2046                    BEGIN
2047                     ERROR(26,1,-1);
2048                     STACK[STKPTR].EXPTYPE := BOOL
2049                    END;
2050                   POPEVALSTACK(OPSTORE[1],SELSTORE[1]);
2051                   NEWLABEL(OPSTORE[2]);
2052                   PUTS('jmpf         ',2,1);
2053                   STACK[STKPTR - 1].DES.SYMNAME := OPSTORE[2]
2054                  END;
2055
2056                39 :      (* <WHILE DO> ::= <WHILE HEAD>DO<STMT GP>END WHILE *)
2057                  BEGIN
2058                   OPSTORE[1] := STACK[STKPTR -4].DES.SYMNAME;
2059                   OPSTORE[2] := STACK[STKPTR -4].DES.TMPNAME;
2060                   PUTS('whend       ',2,0)
2061                  END;
2062
2063                40 :    (* <WHILE HEAD> ::= <WHILE><EXPRESSION>:<MAXLOOPCOUNT> *)
2064                  BEGIN
2065                   POPEVALSTACK(OPSTORE[1],SELSTORE[1]);
2066                   NEWLABEL(OPSTORE[2]);
```

```
2067                    STACK[STKPTR - 3].DES.TMPNAME := OPSTORE[2];
2068                    SELSTORE[1] := STACK[STKPTR].DES.INTVAL;
2069                    PUTS('whilecon  ',2,1)
2070                  END;
2071
2072                41 :                (* <WHILE> ::= WHILE *)
2073                  BEGIN
2074                   NEWLABEL(OPSTORE[1]);
2075                   STACK[STKPTR].DES.SYMNAME := OPSTORE[1];
2076                   PUTS('whilestart',1,0)
2077                  END;
2078
2079                42 :      (* <FOR LOOP> ::= <FOR HEAD>DO<STMT GP>END FOR *)
2080                  WITH STACK[STKPTR - 4] DO
2081                   BEGIN
2082                    OPSTORE[1] := DES.SYMLOC.SYMNAME;
2083                    SELSTORE[1] := DES.SYMLOC.PRECISION3;
2084                    OPSTORE[2] := DES.SYMNAME;
2085                    OPSTORE[3] := DES.TMPNAME;
2086                    SELSTORE[2] := DES.INTVAL;
2087                    PUTS('forend    ',3,2)
2088                   END;
2089
2090                43 :(* <FORHEAD> ::= FOR *ID* FROM <EXPRESSION> TO *)
2091                    (* <EXPRESSION> : <MAXLOOPCOUNT> *)
2092                  BEGIN
2093                   IF STACK[STKPTR - 6].DES.SYMLOC.KIND <> ARITHMETIC THEN
2094                    ERROR(29,1,-1);
2095                   OPSTORE[1] := STACK[STKPTR - 6].DES.SYMNAME;
2096                   SELSTORE[1] := STACK[STKPTR - 6].DES.SYMLOC.PRECISION3;
2097                   POPEVALSTACK(OPSTORE[3],SELSTORE[3]);
2098                   POPEVALSTACK(OPSTORE[2],SELSTORE[2]);
2099                   NEWLABEL(OPSTORE[4]);
2100                   NEWLABEL(OPSTORE[5]);
2101                   SELSTORE[4] := STACK[STKPTR].DES.INTVAL;
2102                   PUTS('forcons   ',5,4);
2103                   STACK[STKPTR - 7].DES.SYMNAME := OPSTORE[4];
2104                   STACK[STKPTR - 7].DES.TMPNAME := OPSTORE[5];
2105                   STACK[STKPTR - 7].DES.SYMLOC := STACK[STKPTR - 6].DES.SYMLOC;
2106                   STACK[STKPTR - 7].DES.INTVAL := SELSTORE[4]
2107                  END;
2108
2109                44 :                    (* <PERFORM TASK> ::= *ID* *)
2110                  BEGIN
2111                   IF (STACK[STKPTR].DES.SYMLOC.KIND <> TASK) AND
2112                      (STACK[STKPTR].DES.SYMLOC.KIND <> FNCTN) THEN
2113                    ERROR(24,1,-1);
2114                   OPSTORE[1] := STACK[STKPTR].DES.SYMNAME;
2115                   PUTS('call      ',1,0)
2116                  END;
2117
2118                45 : (* <PERFORM TASK> ::= <ID> ( <EXPR LIST> : <ID LIST> ) *)
2119                  BEGIN
```

```
2120                    ERROR(36,1,-1);
2121                    IF (STACK[STKPTR - 5].DES.SYMLOC.KIND <> TASK) AND
2122                       (STACK[STKPTR - 5].DES.SYMLOC.KIND <> FNCTN) THEN
2123                     ERROR(24,1,STACK[STKPTR -4].DES.LINEPOS - CC - 3);
2124                    OPSTORE[1] := STACK[STKPTR].DES.SYMNAME;
2125                    PUTS('call      ',1,0)
2126                  END;
2127
2128                46 :              (* <MAXLOOPCOUNT> ::= <NUMBER> *)
2129                  IF STACK[STKPTR].DES.CHARVAL <> '@' (*INTEGER*) THEN
2130                   ERROR(13,1,-1);
2131
2132                47 :              (* <LEFT PART LIST> ::= <NAME> := *)
2133                  BEGIN
2134                   TEMPLIST[1] := STACK[STKPTR - 1].DES.SYMLOC;
2135                   TLI := 1
2136                  END;
2137
2138                48 :  (* <LEFT PART LIST> ::= <LEFT PART LIST> <NAME> := *)
2139                  BEGIN
2140                   TLI := TLI + 1;
2141                   TEMPLIST[TLI] := STACK[STKPTR - 1].DES.SYMLOC
2142                  END;
2143
2144                49 : (* <ASSIGNMENT STATEMENT> ::= <LEFT PART LIST><EXPRESSION>*)
2145                  BEGIN
2146                   POPEVALSTACK(OPSTORE[2],SELSTORE[2]);
2147                   FOR I := 1 TO TLI DO
2148                    BEGIN
2149                     OPSTORE[1] := TEMPLIST[I].SYMNAME;
2150                     WITH TEMPLIST[I] DO
2151                      CASE KIND OF
2152                        BINARY: SELSTORE[1] := PRECISION2;
2153                        ARITHMETIC: SELSTORE[1] := PRECISION3;
2154                        TRANSDEC: SELSTORE[1] := PRECISION6;
2155                        FNCTN: SELSTORE[1] := PRECISION8;
2156                       OTHERWISE BEGIN
2157                        ERROR(32,1,-1);
2158                        SELSTORE[1] := 8
2159                       END
2160                      END
2161                    END;
2162                    PUTS('assign    ',2,2)
2163                   END;
2164
2165                 50 :                 (* <DATA INPUT> ::= SENSE ( <NAME> ) *)
2166                   WITH STACK[STKPTR - 1].DES.SYMLOC DO
2167                    CASE KIND OF
2168                      TRANSDEC : IF TYPE6.SYMNAME <> 'INPUT      ' THEN
2169                                  ERROR(20,1,2)
2170                                 ELSE BEGIN
2171                                  OPSTORE[1] := SYMNAME;
2172                                  SELSTORE[1] := PRECISION6;
```

```
2173                      PUTS('sensecond ',1,1)
2174                     END;
2175          UNDEFINED : ERROR(28,1,-2);
2176          OTHERWISE ERROR(20,1,-2)
2177       END;
2178
2179    51 :                    (* <DATA OUTPUT> ::= ISSUE (<NAME>) *)
2180      WITH STACK[STKPTR - 1].DES.SYMLOC DO
2181              CASE KIND OF
2182               TRANSDEC : IF TYPE6.SYMNAME <> 'OUTPUT     ' THEN
2183                           ERROR(20,1,-2)
2184                          ELSE BEGIN
2185                           OPSTORE[1] := SYMNAME;
2186                           SELSTORE[1] := PRECISION6;
2187                           PUTS('issuevent ',1,1)
2188                          END;
2189               OTHERWISE ERROR(20,1,-2)
2190             END;
2191
2192    52,53,54,55,56,57 :     (* <TIME MEASURE> ::= H/M/S/MS/US/NS *)
2193      STACK[STKPTR].DES.INTVAL := PRODUCTION;
2194
2195    58 :                (* <PERIOD> ::= <NUMBER><TIME MEASURE> *)
2196      BEGIN
2197       IF STACK[STKPTR - 1].DES.CHARVAL = '@' THEN
2198        STACK[STKPTR-1].DES.REALVAL := STACK[STKPTR-1].DES.INTVAL;
2199       WITH STACK[STKPTR - 1] DO
2200         CASE STACK[STKPTR].DES.INTVAL OF
2201              (* CONVERT ALL TIMES TO MILLISECONDS *)
2202           52 : (*HOURS*) DES.REALVAL := DES.REALVAL * 3600000;
2203           53 : (*MINUTES*) DES.REALVAL := DES.REALVAL * 60000;
2204           54 : (*SECONDS*) DES.REALVAL := DES.REALVAL * 1000;
2205           55 : (*MILLISECONDS*);
2206           56 : (*MICROSECONDS*) DES.REALVAL := DES.REALVAL/1000;
2207           57 : (* NANOSECONDS*) DES.REALVAL := DES.REALVAL/1000000
2208         END
2209      END;
2210
2211    59 :                 (* <TIME> ::= <PERIOD> *)  ;
2212
2213    60 :                (* <TIME> ::= <TIME><PERIOD> *)
2214      STACK[STKPTR-1].DES.REALVAL := STACK[STKPTR-1].DES.REALVAL +
2215           STACK[STKPTR].DES.REALVAL;
2216
2217    61 :(* <TIMED BLOCK> ::= <TIMEDBLOCKHEAD> ; <STMT GP>END IN *)
2218      PUTS('ni         ',0,0);
2219
2220    62 :             (* <TIMEDBLCKHEAD> ::= IN <PERIOD> *)
2221      BEGIN
2222       SELSTORE[1] := TRUNC(STACK[STKPTR].DES.REALVAL);
2223       PUTS('in         ',0,1)
2224      END;
2225
```

```
2226                63 :            (* <WAIT> ::= WAIT <PERIOD> *)
2227                  BEGIN
2228                   SELSTORE[1] := TRUNC(STACK[STKPTR].DES.REALVAL);
2229                   PUTS('fixedwait ',0,1)
2230                  END;
2231
2232                64 :            (* <WAIT> ::= WAIT <EXPRESSION> : <PERIOD> *)
2233                  BEGIN
2234                   POPEVALSTACK(OPSTORE[1],SELSTORE[1]);
2235                   SELSTORE[2] := TRUNC(STACK[STKPTR].DES.REALVAL);
2236                   PUTS('waitleast ',1,2);
2237                   IF STACK[STKPTR - 2].EXPTYPE <> INT THEN
2238                     ERROR(38,1,STACK[STKPTR - 1].DES.LINEPOS - CC - 3)
2239                  END;
2240
2241                65 : (* <WAIT UNTIL> ::= <WAITHEAD><EXPRESSION> : <PERIOD> *)
2242                  BEGIN
2243                   POPEVALSTACK(OPSTORE[1],SELSTORE[1]);
2244                   OPSTORE[2] := STACK[STKPTR - 3].DES.SYMNAME;
2245                   NEWLABEL(OPSTORE[3]);
2246                   SELSTORE[2] := TRUNC(STACK[STKPTR].DES.REALVAL);
2247                   PUTS('boolwait  ',3,2);
2248                   IF STACK[STKPTR - 2].EXPTYPE <> BOOL THEN
2249                    ERROR(26,1,STACK[STKPTR - 1].DES.LINEPOS - CC - 3)
2250                  END;
2251
2252                66 :             (* <WAITHEAD> ::= WAIT UNTIL *)
2253                  BEGIN
2254                   NEWLABEL(OPSTORE[1]);
2255                   PUTS('stboolwait',1,0);
2256                   STACK[STKPTR - 1].DES.SYMNAME := OPSTORE[1]
2257                  END
2258
2259                OTHERWISE SEMANTIC1(PRODUCTION) ;
2260               END (*CASE*)
2261              END; (*SEMANTIC*)
2262
2263
2264              PROCEDURE SEMANTIC1 (*PRODUCTION : INTEGER *);
2265                VAR PTR : SYMPTR;
2266                    TYPVAR : EXPTYPES;
2267                    TEMPNAME : ALFA;
2268                    I : INTEGER;
2269
2270                BEGIN
2271                 CASE PRODUCTION OF
2272
2273                  86 :(*<INPUT SPEC> ::= INPUT : <TRANSMISSION BODY> END INPUT*)
2274                     BEGIN
2275
2276                      FOR I := 1 TO TLI DO
2277                       BEGIN
2278                        TEMPLIST[I].TYPE6 := STACK[STKPTR - 4].DES.SYMLOC;
```

```
2279                    OPSTORE[1] := TEMPLIST[I].SYMNAME;
2280                    OPSTORE[2] := TEMPLIST[I].TECHNOLOGY6.SYMNAME;
2281                    SELSTORE[1] := TEMPLIST[I].PRECISION6;
2282                    PUTS('inputport ',2,1);
2283                    PUTSYM('INPUTPORT',2,1)
2284                   END;
2285                  TLI := 0
2286                 END;
2287
2288               87 :(* <OUTPUT SPEC> ::= OUTPUT :      *)
2289                   (* <TRANSMISSION BODY> END OUTPUT *)
2290                 BEGIN  (* ASSIGN TYPE = 'OUTPUT' *)
2291                  FOR I := 1 TO TLI DO
2292                   BEGIN
2293                    TEMPLIST[I].TYPE6 := STACK[STKPTR - 4].DES.SYMLOC;
2294                    OPSTORE[1] := TEMPLIST[I].SYMNAME;
2295                    OPSTORE[2] := TEMPLIST[I].TECHNOLOGY6.SYMNAME;
2296                    SELSTORE[1] := TEMPLIST[I].PRECISION6;
2297                    PUTS('outputport',2,1);
2298                    PUTSYM('OUTPUTPORT',2,1)
2299                   END;
2300                  TLI := 0
2301                 END;
2302
2303               94 :    (* <DUPLEX SPEC> ::= DUPLEX :      *)
2304                       (* <TRANSMISSION BODY> END DUPLEX *)
2305                 BEGIN  (* ASSIGN TYPE = 'DUPLEX' *)
2306                  FOR I := 1 TO TLI DO
2307                   BEGIN
2308                    TEMPLIST[I].TYPE6 := STACK[STKPTR - 4].DES.SYMLOC;
2309                    OPSTORE[1] := TEMPLIST[I].SYMNAME;
2310                    OPSTORE[2] := TEMPLIST[I].TECHNOLOGY6.SYMNAME;
2311                    SELSTORE[1] := TEMPLIST[I].PRECISION6;
2312                    PUTS('in/outport',2,1);
2313                    PUTSYM('IN/OUTPORT',2,1)
2314                   END;
2315                  TLI := 0
2316                 END;
2317
2318               95 :(* <BINARY SPEC> ::= BINARY : <BINARY BODY> END BINARY *);
2319
2320               96 :      (* <ARITHMETIC SPEC> ::=
2321                           ARITHMETIC : <ARITHMETIC BODY> END ARITHMETIC *);
2322
2323               99 :      (* <TRANSMISSION DEC> ::= <ID> , <BINARY PRECISION> ,
2324                                 <TECHNOLOGY> *)
2325                 BEGIN
2326                  PTR := STACK[STKPTR - 4].DES.SYMLOC;
2327                  IF PTR.KIND <> UNDEFINED THEN
2328                   ERROR(14,1,STACK[STKPTR - 3].DES.INTVAL - 1);
2329                  PTR.KIND := TRANSDEC;
2330                  PTR.PRECISION6 := STACK[STKPTR - 2].DES.INTVAL;
2331                  PTR.TECHNOLOGY6 := STACK[STKPTR].DES.SYMLOC
```

```
2332                    END;
2333
2334                  102 :   (* <BINARY DEC> ::= <ID><STRUCTURE> , *)
2335                          (* <BINARY PRECISION> <INITIAL VALUE> *)
2336                    BEGIN
2337                     PTR := STACK[STKPTR - 4].DES.SYMLOC;
2338                     IF PTR.KIND <> UNDEFINED THEN
2339                      ERROR(14,1,0);
2340                     PTR.KIND := BINARY;
2341                     PTR.PRECISION2 := STACK[STKPTR - 1].DES.INTVAL;
2342                     IF STACK[STKPTR].DES.CHARVAL = '@' THEN
2343                      PTR.IVAL2 := 0
2344                     ELSE
2345                      PTR.IVAL2 := STACK[STKPTR].DES.INTVAL
2346                    END;
2347
2348                  105 :     (* <ARITHMETIC DEC> ::= <ID><STRUCTURE> ,*)
2349                            (* <DECIMAL PRECISION> <INITIAL VALUE>   *)
2350                    BEGIN
2351                     PTR := STACK[STKPTR - 4].DES.SYMLOC;
2352                     IF PTR.KIND <> UNDEFINED THEN
2353                      ERROR(14,1,0);
2354                     PTR.KIND := ARITHMETIC;
2355                     PTR.PRECISION3 := STACK[STKPTR - 1].DES.INTVAL;
2356                     IF STACK[STKPTR].DES.CHARVAL = '@' THEN
2357                      PTR.IVAL3 := 0
2358                     ELSE
2359                      PTR.IVAL3 := STACK[STKPTR].DES.INTVAL;
2360                     OPSTORE[1] := PTR.SYMNAME;
2361                     SELSTORE[1] := PTR.PRECISION3;
2362                     SELSTORE[2] := PTR.IVAL3;
2363                     PUTS('var       ',1,2);
2364                     PUTSYM('VARIABLE  ',1,2)
2365                    END;
2366
2367                  106 :          (* <STRUCTURE> ::=     *);
2368
2369                  107 :         (* <STRUCTURE> ::= ( <NUMBER LIST> )  *)
2370                     ERROR(36,1,-1);
2371
2372                  108 ,         (* <NUMBER LIST> ::= *NUMBER* *)
2373
2374                  109 :      (* <NUMBER LIST> ::= <NUMBER LIST> , *NUMBER* *)
2375                     BEGIN
2376                      IF STACK[STKPTR].DES.CHARVAL <> '@' THEN
2377                       ERROR(18,1,-1);
2378                      NLI := NLI + 1;
2379                      SELSTORE[NLI] := STACK[STKPTR].DES.INTVAL;
2380                      IF PRODUCTION = 111 THEN
2381                       STACK[STKPTR].DES.INTVAL := NLI
2382                      ELSE
2383                       STACK[STKPTR - 2].DES.INTVAL := NLI
2384                     END;
```

```
2385
2386                    110 :       (* <BINARY PRECISION> ::= <NUMBER>  *)
2387                       IF STACK[STKPTR].DES.CHARVAL <> '@' THEN
2388                        ERROR(13,1,-1);
2389
2390                    111 :       (* <DECIMAL PRECISION> ::= <NUMBER> *)
2391                       IF STACK[STKPTR].DES.CHARVAL <> '@' THEN
2392                        ERROR(18,1,-1);
2393
2394                    112 :        (* <INITIAL VALUE> *)
2395                       STACK[STKPTR + 1].DES.CHARVAL := '@';
2396
2397                    113 :        (* <INITIAL VALUE> ::= <NUMBER>  *)
2398                       IF STACK[STKPTR].DES.CHARVAL = '@' THEN
2399                        BEGIN
2400                         STACK[STKPTR - 1].DES.CHARVAL := 'I';
2401                         STACK[STKPTR - 1].DES.INTVAL := STACK[STKPTR].DES.INTVAL
2402                        END
2403                       ELSE
2404                        ERROR(18,1,-1);
2405
2406                    114,115,116 :      (* <TECHNOLOGY> ::= TTL/ECL/IIL *);
2407
2408                    121 : (* <CODE SPEC> ::= CODE : *ID* , <BINARY PRECISION> ;
2409                                          <CHARACTER REP LIST> END CODE *)
2410                       BEGIN
2411                        FOR I := 1 TO TLI DO
2412                         TEMPLIST[I].CODID5 := STACK[STKPTR - 6].DES.SYMLOC;
2413                        TLI := 0
2414                       END;
2415
2416                    97,98,118,119,122,123 :   (*  <A> ::= <B> , <A> ::= <A> <B> *)
2417                      BEGIN
2418                        TLI := TLI + 1;
2419                        TEMPLIST[TLI] := STACK[STKPTR - 1].DES.SYMLOC
2420                       END;
2421
2422                    125,126,127,128,129 :(* <CODE ID> ::= <ID>/ASCII6/
2423                                          ASCII7/EBCIDIC/BCD *);
2424
2425                    130 :       (* <ID LIST> ::=      *)
2426                       TLI := 0;
2427
2428                    131 :       (* <IDLIST> ::= <ID> *)
2429                       BEGIN
2430                        TEMPLIST[1] := STACK[STKPTR].DES.SYMLOC;
2431                        TLI := 1
2432                       END;
2433
2434                    132 :       (* <ID LIST> ::= <IDLIST> , <ID> *)
2435                       BEGIN
2436                        TLI := TLI + 1;
2437                        TEMPLIST[TLI] := STACK[STKPTR].DES.SYMLOC
```

```
2438                    END;
2439
2440                 133 :       (* <NAME> ::= <ID> *);
2441
2442                 134 :       (* <NAME> ::= <ID> ( <EXPRLIST> ) *)
2443                    BEGIN
2444                     STACK[STKPTR - 3].DES.CHARVAL := 'A' ; (*ARRAY*)
2445                     ERROR(36,1,-1)
2446                    END;
2447
2448                 135 :
2449                    BEGIN
2450                     STACK[STKPTR - 5].DES.CHARVAL := 'B'; (*BIT FIELD*)
2451                     ERROR(37,1,-1)
2452                    END;
2453
2454                 136 :   (* <FORMAL PARAM LIST> ::=     *)
2455                    FIRSTPARAM := NIL;
2456
2457                 137 :   (* <FORMAL PARAM LIST> ::= ( <ID LIST> : <ID LIST> ) *)
2458                    ERROR(33,1,-1);
2459
2460                 138 :     (* <PROC> ::= <TASK>  *);
2461
2462                 139 :     (* <PROC> ::= <FUNCTION> *);
2463
2464                 140,       (* <TASK> ::= <TASK HEAD> ; <ZOPT PROC DEC GP>
2465                                    <STMT GP> END <ID> *)
2466
2467                 146 : (* <FUNCTION> ::= <FUNCTION HEAD> ; <ZOPT PROC DEC GP>
2468                                    <STMT GP> END <ID> *)
2469                    BEGIN
2470                     IF STACK[STKPTR - 5].DES.SYMLOC <> STACK[STKPTR].DES.SYMLOC
2471                      THEN ERROR(25,1,-1);
2472                     OPSTORE[1] := STACK[STKPTR - 5].DES.SYMNAME;
2473                     PUTS('exitproc  ',1,0)
2474                    END;
2475
2476                 147,  (* <FUNCTION HEAD> ::= FUNCTION *ID* <FORMAL PARAM LIST>
2477                               BINARY , <BINARY PRECISION><INITIAL VALUE> *)
2478
2479                 148 : (* <FUNCTION HEAD> ::= FUNCTION *ID* <FORMAL PARAM LIST>
2480                          ARITHMETIC , <DECIMAL PRECISION><INITIAL VALUE> *)
2481                    BEGIN
2482                     PUTT(STACK[STKPTR - 6].DES.SYMNAME);
2483                     OPSTORE[1] := STACK[STKPTR - 6].DES.SYMNAME;
2484                     PUTS('proc      ',1,0);
2485                     WITH STACK[STKPTR - 6].DES.SYMLOC DO
2486                      BEGIN
2487                       KIND := FNCTN;
2488                       PARAMLIST8 := FIRSTPARAM;
2489                       TYPE8 := STACK[STKPTR - 3].DES.SYMLOC;
2490                       PRECISION8 := STACK[STKPTR - 1].DES.INTVAL;
```

```
2491                      IVAL8 := STACK[STKPTR].DES.INTVAL
2492                     END;
2493                    STACK[STKPTR - 7] := STACK[STKPTR - 6]
2494                   END;
2495
2496                 145 :       (* <TASK HEAD> ::= TASK *ID* <FORMAL PARAM LIST> *)
2497                   BEGIN
2498                    PUTT(STACK[STKPTR - 1].DES.SYMNAME);
2499                    OPSTORE[1] := STACK[STKPTR - 1].DES.SYMNAME;
2500                    PUTS('proc      ',1,0);
2501                    WITH STACK[STKPTR - 1].DES.SYMLOC DO
2502                     BEGIN
2503                      KIND := TASK;
2504                      PARAMLIST7 := FIRSTPARAM
2505                     END;
2506                    STACK[STKPTR - 2] := STACK[STKPTR - 1]
2507                   END;
2508
2509                 156 :     (* <RANK> ::= <NU> *)
2510                   STACK[STKPTR].DES.LINEPOS := STACK[STKPTR].DES.INTVAL;
2511
2512                 157 :     (* <RANK> ::= <NU> . <PI> *)
2513                   BEGIN
2514                    STACK[STKPTR - 2].DES.LINEPOS :=
2515                              STACK[STKPTR - 2].DES.INTVAL;
2516                    STACK[STKPTR - 2].DES.INTVAL := STACK[STKPTR].DES.INTVAL
2517                   END;
2518
2519                 158,    (* <NU> ::= <NUMBER> *)
2520                 159 :   (* <PI> ::= <NUMBER> *)
2521                   IF STACK[STKPTR].DES.CHARVAL <> '@' THEN
2522                    ERROR(13,1,-1);
2523
2524                 160 :   (* <QUALIFICATION> ::=     *)
2525                   STACK[STKPTR + 1].DES.CHARVAL := '@';
2526
2527                 161 :     (* <QUALIFICATION> ::= IF <EXPRESSION> *)
2528                   BEGIN
2529                    STACK[STKPTR - 1].DES.CHARVAL := 'Q';
2530                    IF STACK[STKPTR].EXPTYPE <> BOOL THEN
2531                     ERROR(26,1,-1);
2532                    POPEVALSTACK(OPSTORE[1],SELSTORE[1]);
2533                    NEWLABEL(OPSTORE[2]);
2534                    PUTS('jmpf      ',2,1);
2535                    STACK[STKPTR - 1].DES.SYMNAME := OPSTORE[2]
2536                   END;
2537
2538                 162,163,164,165,166 :  (* <EPISODE TIMING> ::=    / : <ROE> /
2539                                         : <ROE> , <B1> / : <ROE> , <B1> , <B2> /
2540                                         : <ROE> , <B1> , <B2> , <RANK> *)
2541                   BEGIN
2542                    FOR I := 1 TO 5 DO
2543                     SELSTORE[I] := 0;
```

```
2544                    CASE PRODUCTION OF
2545                     162 : ;
2546                     163 : SELSTORE[1] := STACK[STKPTR].DES.INTVAL;
2547                     164 : BEGIN
2548                            SELSTORE[1] := STACK[STKPTR - 2].DES.INTVAL;
2549                            SELSTORE[2] := STACK[STKPTR].DES.INTVAL
2550                           END;
2551                     165 : BEGIN
2552                            SELSTORE[1] := STACK[STKPTR - 4].DES.INTVAL;
2553                            SELSTORE[2] := STACK[STKPTR - 2].DES.INTVAL;
2554                            SELSTORE[3] := STACK[STKPTR].DES.INTVAL
2555                           END;
2556                     166 : BEGIN
2557                            SELSTORE[1] := STACK[STKPTR - 6].DES.INTVAL;
2558                            SELSTORE[2] := STACK[STKPTR - 4].DES.INTVAL;
2559                            SELSTORE[3] := STACK[STKPTR - 2].DES.INTVAL;
2560                            SELSTORE[4] := STACK[STKPTR].DES.LINEPOS;
2561                            SELSTORE[5] := STACK[STKPTR].DES.INTVAL
2562                           END
2563                          END
2564                  END;
2565
2566                167 :   (*<WHEN DO> ::= <QUALIFICATION> WHEN <NAME>
2567                                        <EPISODE TIMING> DO <TASK LIST> *)
2568
2569                   BEGIN
2570                    WITH STACK[STKPTR - 3].DES.SYMLOC DO
2571                     CASE KIND OF
2572                     UNDEFINED : ERROR(28,1,STACK[STKPTR -4].DES.LINEPOS-CC+2);
2573                      BINARY : ;
2574                      FNCTN : IF TYPE8.SYMNAME <> 'BINARY    ' THEN
2575                                ERROR(30,1,STACK[STKPTR-4].DES.LINEPOS-CC+2);
2576                      OTHERWISE ERROR(30,1,STACK[STKPTR-4].DES.LINEPOS-CC)
2577                     END;
2578                    FOR I := 1 TO OPI DO
2579                     PUTA(STACK[STKPTR-3].DES.SYMNAME,OPSTORE[I],5);
2580                    IF STACK[STKPTR-5].DES.CHARVAL = 'Q' THEN
2581                     BEGIN
2582                      OPSTORE[1] := STACK[STKPTR-5].DES.SYMNAME;
2583                      PUTS('loc       ',1,0);
2584                      PUTSYM('LOC       ',1,0)
2585                     END
2586                   END;
2587
2588                168 : (* <SIMPLE DO> ::= <QUALIFICATION> DO <TASK LIST> <RANK>*)
2589
2590                   BEGIN
2591                    SELSTORE[1] := 0;
2592                    FOR I := 1 TO OPI DO
2593                     PUTA('          ',OPSTORE[I],5);
2594                    IF STACK[STKPTR-3].DES.CHARVAL = 'Q' THEN
2595                     BEGIN
```

```
2596                       OPSTORE[1] := STACK[STKPTR-5].DES.SYMNAME;
2597                       PUTS('loc       ',1,0)
2598                      END
2599                    END;
2600
2601                 169,    (* <EVERY> ::= <QUALIFICATION> EVERY <ROE>
2602                                        DO <TASK LIST> *)
2603
2604                 170 :   (* <AT TIME> ::= <QUALIFICATION> AT <TIME>
2605                                        DO <TASK LIST> *)
2606
2607                    BEGIN
2608                     SELSTORE[1] := STACK[STKPTR-2].DES.INTVAL;
2609                     FOR I := 1 TO OPI DO
2610                      PUTA('          ',OPSTORE[I],5);
2611                     IF STACK[STKPTR - 3].DES.CHARVAL = 'Q' THEN
2612                      BEGIN
2613                       OPSTORE[1] := STACK[STKPTR - 5].DES.SYMNAME;
2614                       PUTS('loc       ',1,0)
2615                      END
2616                    END;
2617
2618
2619                 171 :  (* <TASK LIST> ::= <NAME> *)
2620
2621                    BEGIN
2622                     WITH STACK[STKPTR].DES.SYMLOC DO
2623                      CASE KIND OF
2624                       UNDEFINED : ERROR(28,1,-1);
2625                       TASK : ;
2626                       FNCTN : ;
2627                       OTHERWISE ERROR(34,1,-1)
2628                      END;
2629                     OPSTORE[1] :=STACK[STKPTR].DES.SYMNAME;
2630                     OPI := 1
2631                    END;
2632
2633                 172 :  (* <TASK LIST> ::= <TASK LIST> THEN <NAME> *)
2634
2635                    BEGIN
2636                     WITH STACK[STKPTR].DES.SYMLOC DO
2637                      CASE KIND OF
2638                       UNDEFINED : ERROR(28,1,-1);
2639                       TASK : ;
2640                       FNCTN : ;
2641                       OTHERWISE ERROR(34,1,-1)
2642                      END;
2643                     OPI := OPI + 1;
2644                     OPSTORE[OPI] := STACK[STKPTR].DES.SYMNAME
2645                    END;
2646
2647                 182 :  (* <DESIGN CRITERIA> ::= DESIGN CRITERIA
2648                           METRIC <METRIC> ; VOLUMES <NUMBER LIST> ;
```

```
2649                        MONITORS <NUMBER LIST> ; *)
2650
2651                    BEGIN
2652                     PUTD(STACK[STKPTR-7].DES.SYMNAME,STACK[STKPTR-4].DES.INTVAL,
2653                          STACK[STKPTR-1].DES.INTVAL-STACK[STKPTR-4].DES.INTVAL);
2654                     NLI := 0
2655                    END;
2656
2657                183,184,185 :  (* <METIRC> ::= FIRST/COST/POWER *);
2658
2659                190 : (* PRINT ALL CONSTANTS USED DURING COMPILATION *)
2660
2661                   BEGIN
2662                    PUTT('  SYSTEM  ');
2663                    FOR I := 1 TO CSI DO
2664                     BEGIN
2665                      OPSTORE[1] := CONSTANTSTORE[I].NAME;
2666                      SELSTORE[1] := CONSTANTSTORE[I].VAL;
2667                      SELSTORE[2] := CONSTANTSTORE[I].PRECISION;
2668                      PUTS('cons       ',1,2);
2669                      PUTSYM('CONS        ',1,2)
2670                     END;
2671                    PRINTEMPS
2672                   END;
2673
2674                END   (*CASE PRODUCTION OF *)
2675                END;  (* PROCEDURE SEMANTIC1 *)
2676
2677             BEGIN (*MAIN*)
2678
2679              INITIALIZE;
2680              PUTT('   SYSTEM   ');
2681              PUTS('MAIN      ',0,0);
2682              PARSE;
2683
2684              (*THE BELOW STMTS CHECK TO SEE IF TOGGLES HAVE*)
2685              (*BEEN TURNED ON TO TRACE THE PARSE EXECUTION *)
2686              (*AND CALLS THE PROCEDURE TO PRINT THE DETAILS*)
2687
2688              99: IF LINERRPTR <> 0 THEN PRINTLINERRORS;
2689                  IF PROGRAMERRFLAG THEN PRINTERRORS;
2690                  IF SWITCH[PRINTTABLE] THEN GOPRINTTABLE
2691             END.
```

```
CSDL TRANSLATOR                                    PAGE 1
NAVAL POSTGRADUATE SCHOOL
18-APR-19811:17:03.1
    1 -- PRINTTABLE
    2
    3 IDENTIFICATION
    4
    5    DESIGNER : "ALAN ROSS"
    6    DATE : "12-28-83"
    7    PROJECT : "DUAL PROCESS CONTROL APPLICATION"
    8
    9 DESIGN CRITERIA
   10
   11   METRIC FIRST ;
   12   VOLUMES 8 ;
   13   MONITORS 8 ;
   14
   15
   16 ENVIRONMENT
   17
   18    INPUT : CONSIN,8,TTL ; CONST,8,TTL ; FLGA,1,TTL ;
   19            PINA,8,TTL ; FLGB,1,TTL ;
   20            PINB,8,TTL ; END INPUT ;
   21
   22    OUTPUT : VA,8,TTL ; VB,8,TTL ; END OUTPUT ;
   23
   24    ARITHMETIC : KCA,8 ; KCB,8 ; CNTB,8 ; ITIA,8 ; ITIB,8 ; AINT,8 ; TDA,8 ;
   25           TDB,8 ; BINT,8 ; VSA,8 ; VSB,8 ; BDIFF,8 ; PSA,8 ; PSB,8 ; CONPTT,8 ;
   26           EA,8 ; EB,8 ; KPIA,8 ; EA1,8 ; EA2,8 ; EB1,8 ; EB2,8 ; KPIB,8 ;
   27           END ARITHMETIC ;
   28
   29
   30 PROCEDURES
   31
   32    FUNCTION DATAA:
   33        BINARY ,1;
   34        SENSE (FLGA);
   35        IF FLGA = 1 THEN DATAA := 1; END IF ;
   36    END DATAA ;
   37
   38     FUNCTION DATAB:
   39        BINARY ,1;
   40        SENSE (FLGB);
   41        IF FLGB = 1 THEN DATAB := 1; END IF ;
   42     END DATAB ;
   43
   44    FUNCTION BCNT :
   45        BINARY ,1 ;
   46        IF CNTB >= 4 THEN BCNT := 1; END IF ;
   47    END BCNT ;
   48
   49    TASK AFIX ;
   50        ARITHMETIC : ADIFF,8 ; END ARITHMETIC ;
```

```
   51        SENSE (PINA);
   52        EA := PINA*KCA - PSA;
   53        ADIFF := (3*EA - 4*EA1 + EA2) *5;
   54        AINT := AINT + EA/KCA;
   55        VA := VSA + KCA*(EA + ITIA*AINT + TDA*ADIFF);
   56        ISSUE (VA);
   57        DATAA := 0;
   58        EA2 := EA1;
   59        EA1 := EA;
   60     END AFIX;
   61
   62     TASK BCALC;
   63        SENSE (PINB);
   64        EB := PINB*KCB - PSB;
   65        BDIFF := (3*EB - 4*EB1 + EB2)*10;
   66        BINT := BINT + EB/KCB;
   67        CNTB := CNTB + 1;
   68        DATAB := 0;
   69     END BCALC;
   70
   71     TASK BFIX;
   72        CNTB := 0;
   73        VB := VSB + KCB*(EB + ITIB*BINT + TDB*BDIFF);
   74        ISSUE (VB);
   75     END BFIX;
   76
   77     FUNCTION CONFLG:
   78        BINARY,1;
   79        SENSE (CONSIN);
   80        IF CONSIN > 0 THEN CONFLG := 0; END IF;
   81     END CONFLG;
   82
   83     TASK CHGCON;
   84        SENSE (CONST);
   85        IF CONPTT = 1 THEN KCA := CONST; END IF;
   86        IF CONPTT = 2 THEN ITIA := 1/CONST; END IF;
   87        IF CONPTT = 3 THEN TDA := CONST; END IF;
   88        IF CONPTT = 4 THEN VSA := CONST; END IF;
   89        IF CONPTT = 5 THEN PSA := CONST; END IF;
   90        IF CONPTT = 6 THEN AINT := CONST; END IF;
   91        IF CONPTT = 7 THEN KCB := CONST; END IF;
   92        IF CONPTT = 8 THEN ITIB := 1/CONST; END IF;
   93        IF CONPTT = 9 THEN TDB := CONST; END IF;
   94        IF CONPTT = 10 THEN VSB := CONST; END IF;
   95        IF CONPTT = 11 THEN PSB := CONST; END IF;
   96        IF CONPTT = 12 THEN BINT := CONST; END IF;
   97     END CHGCON;
   98
   99
```

```
  100 CONTINGENCY LIST
1
CSDL TRANSLATOR                                                PAGE 3
NAVAL POSTGRADUATE SCHOOL
18-APR-19811:17:03.1
  101        WHEN DATAA:100 MS DO AFIX;
  102        WHEN DATAB: 50 MS DO BCALC;
  103        WHEN BCNT: 100 MS DO BFIX;
  104        WHEN CONFLG DO CHGCON;
  105
  106     END
    1
     FUNCTION      RESWD          48
    2
     DATAB         FUNCTION    BINARY          1
     M             RESWD          57
    3
     DESIGN        RESWD          36
    4
    5
    6
     THEN          RESWD          72
    7
     PROJECT       RESWD          67
    8
    9
     BCNT          FUNCTION    BINARY          1
     TERM          RESWD          71
   10
     DATAA         FUNCTION    BINARY          1
     FOR           RESWD          46
     ARITHMETIC    RESWD          25
   11
     AND           RESWD          24
   12
     PRINTTABLE    UNDEFINED
   13
     EA            ARITHMETIC      8     0
     VSB           ARITHMETIC      8     0
   14
     EBCIDIC       RESWD          40
     CODE          RESWD          31
   15
     CHGCON        TASK
     VSA           ARITHMETIC      8     0
     VOLUMES       RESWD          78
     MS            RESWD          60
   16
   17
     UNTIL         RESWD          75
   18
     KCB           ARITHMETIC      8     0
     TTL           RESWD          74
```

```
  TASK          RESWD         70
  POWER         RESWD         65
  ISSUE         RESWD         55
19
  KCA           ARITHMETIC     8     0
  AT            RESWD         28
20
  BINT          ARITHMETIC     8     0
  PINB          TRANSDEC INPUT      TTL           8
  PINA          TRANSDEC INPUT      TTL           8
  WHILE         RESWD         81
  CONTINGENC    RESWD         32
21
  TDB           ARITHMETIC     8     0
  WHEN          RESWD         80
  EVERY         RESWD         44
  BINARY        RESWD         30
22
  TDA           ARITHMETIC     8     0
23
  BDIFF         ARITHMETIC     8     0
  IIL           RESWD         52
  BCD           RESWD         29
24
  EB            ARITHMETIC     8     0
25
  CONSIN        TRANSDEC INPUT      TTL           8
  DATE          RESWD         35
26
  AFIX          TASK
  TO            RESWD         73
27
  ECL           RESWD         41
28
  VA            TRANSDEC OUTPUT     TTL           8
  LIST          RESWD         56
29
30
  NOT           RESWD         61
31
  METRIC        RESWD         58
  IN            RESWD         53
32
  IDENTIFICA    RESWD         50
33
  S             RESWD         68
34
  DO            RESWD         38
35
  CNTB          ARITHMETIC     8     0
  MONITORS      RESWD         59
36
  FIRST         RESWD         45
```

```
  DUPLEX        RESWD          39
37
  PSB           ARITHMETIC      8     0
  ENVIRONMEN    RESWD          43
  ASCII7        RESWD          27
  ASCII6        RESWD          26
38
  CONPTT        ARITHMETIC      8     0
  PSA           ARITHMETIC      8     0
  ITIB          ARITHMETIC      8     0
  ITIA          ARITHMETIC      8     0
  CONST         TRANSDEC INPUT      TTL            8
  US            RESWD          76
  NS            RESWD          62
39
  VB            TRANSDEC OUTPUT     TTL            8
  OUTPUT        RESWD          64
40
  PROCEDURES    RESWD          66
41
  KPIB          ARITHMETIC      8     0
  KPIA          ARITHMETIC      8     0
  DESIGNER      RESWD          37
42
  EA2           ARITHMETIC      8     0
43
44
  EA1           ARITHMETIC      8     0
45
46
47
  FLGB          TRANSDEC INPUT      TTL            1
  FLGA          TRANSDEC INPUT      TTL            1
  WAIT          RESWD          79
  FROM          RESWD          47
  COST          RESWD          33
48
49
  BFIX          TASK
  BCALC         TASK
  H             RESWD          49
50
  VARIABLES     RESWD          77
  SENSE         RESWD          69
  END           RESWD          42
51
  AINT          ARITHMETIC      8     0
  OR            RESWD          63
52
  IF            RESWD          51
53
  CONFLG        FUNCTION   BINARY         1
  EB2           ARITHMETIC      8     0
```

```
  CRITERIA      RESWD          34
54
  ADIFF         ARITHMETIC      8     0
  EB1           ARITHMETIC      8     0
  INPUT         RESWD          54
```

APPENDIX F

SYMBOL TABLE

```
S. INPUTPORT(CONSIN,TTL:8)
S. INPUTPORT(CONST,TTL:8)
S. INPUTPORT(FLGA,TTL:1)
S. INPUTPORT(PINA,TTL:8)
S. INPUTPORT(FLGB,TTL:1)
S. INPUTPORT(PINB,TTL:8)
S.OUTPUTPORT(VA,TTL:8)
S.OUTPUTPORT(VB,TTL:8)
S.VARIABLE  (KCA:8,0)
S.VARIABLE  (KCB:8,0)
S.VARIABLE  (CNTB:8,0)
S.VARIABLE  (ITIA:8,0)
S.VARIABLE  (ITIB:8,0)
S.VARIABLE  (AINT:8,0)
S.VARIABLE  (TDA:8,0)
S.VARIABLE  (TDB:8,0)
S.VARIABLE  (BINT:8,0)
S.VARIABLE  (VSA:8,0)
S.VARIABLE  (VSB:8,0)
S.VARIABLE  (BDIFF:8,0)
S.VARIABLE  (PSA:8,0)
S.VARIABLE  (PSB:8,0)
S.VARIABLE  (CONPTT:8,0)
S.VARIABLE  (EA:8,0)
S.VARIABLE  (EB:8,0)
S.VARIABLE  (KPIA:8,0)
S.VARIABLE  (EA1:8,0)
S.VARIABLE  (EA2:8,0)
S.VARIABLE  (EB1:8,0)
S.VARIABLE  (EB2:8,0)
S.VARIABLE  (KPIB:8,0)
S.LOC       (@01:)
S.LOC       (@02:)
S.LOC       (@03:)
S.VARIABLE  (ADIFF:8,0)
S.LOC       (@04:)
S.LOC       (@05:)
S.LOC       (@06:)
S.LOC       (@07:)
S.LOC       (@08:)
S.LOC       (@09:)
S.LOC       (@10:)
S.LOC       (@11:)
S.LOC       (@12:)
S.LOC       (@13:)
S.LOC       (@14:)
S.LOC       (@15:)
S.LOC       (@16:)
S.CONS      (@C01:1,8)
S.CONS      (@C02:4,8)
S.CONS      (@C03:3,8)
S.CONS      (@C04:5,8)
S.CONS      (@C05:0,8)
```

```
S.CONS      (@C06:10,8)
S.CONS      (@C07:2,8)
S.CONS      (@C08:6,8)
S.CONS      (@C09:7,8)
S.CONS      (@C10:8,8)
S.CONS      (@C11:9,8)
S.CONS      (@C12:11,8)
S.CONS      (@C13:12,8)
```

LIST OF REFERENCES

1. Samish, F., "With Grand Designs," Micro Decisions(GB), No. 15 37-8, January 1983.

2. Ross, A.A., Computer Aided Design of Microprocessor-Based Controllers, Ph. D. Thesis, University of California, Davis, 1978.

3. Matelin, M.N., "Automating the Design of Dedicated Real Time Control Systems," VCRL-78651, Lawrence Livermore Laboratory, 21 August 1976.

4. Sherlock, B.J., User-Friendly Syntax Directed Input to a Computer Aided Design System, M.S. Thesis, Naval Postgraduate School, Monterey, California, 1983.

5. Walden, H.J., The Application of a General Purpose Data Base Management System to Design Automation, M.S. Thesis, Naval Postgraduate School, Monterey, California, 1983.

6. Barrett, W.A., Couch, J.D., Compiler Construction: Theory and Practice, Science Research Associates Inc., 1979.

7. Chomsky, N., "Three Models for the Description of Language", IEEE Transactions on Information Theory 2, 1956.

8. Shannon, A., The LR System. FORTRAN source listing for the LR system, National Energy Software Center, Version 61, Argonne, Illinois, 1979.

9. Wetherell, C. and Shannon, A., "LR, Automatic Parser Generator and LR(1) Parser.", Lawrence Livermore Laboratory, Livermore, California, 1979.

10. Myers, G.J., Composite Structured Design, Van Nostrand Reinhold company, New York, 1978.

11. Garlington, A.R., Preliminary Design and Implementation of an Ada Pseudo-Machine , M.S. Thesis, Air Force Institute of Technology, Dayton, 1981.

12. Dijkstra, E.W., "A Constructive Approach to the Problem of Program Correctness", BIT, Vol. 8, No. 3, 1968.

13. Howden, W.E., "Introduction to the Theory of Testing", Tutorial: Software Testing & Validation Techniques, IEEE Catalog No. EHO 138-8, 1978.

14. Miller, E., "Introduction to Software Testing Technology", Tutorial: Software Testing & Validation Techniques, IEEE Catalog No. EHO 138-8, 1978.

15. Howden, W.E., "Empirical Studies of Software Validation", Tutorial: Software Testing & Validation Techniques, IEEE Catalog No. EHO 138-8, 1978.

16. Matelan, M.N., "Automating the Design of Dedicated Real Time Control Systems", Lawrence Livermore Laboratory, Preprint UCRL-78651, 1976

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Dudley Knox Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
4.	Office of Research Administration Code 012A Naval Postgraduate School Monterey, California 93943	1
5.	Computer Technologies Curricular Office Code 37 Naval Postgraduate School Monterey, California 93943	1
6.	Thomas H. Carson 458 Papaya Court Jacksonville, Florida 32225	2
7.	LtCol. Alan A. Ross, USAF, Code 52Rs Department of Computer Science Naval Postgraduate School Monterey, California 93943	3

208296

Thesis
C27265 Carson
c.1 An input translator
for a computer-aided
design system.

208296

Thesis
C27265 Carson
c.1 An input translator
for a computer-aided
design system.

thesC27265
An input translator for a computer-aided
3 2768 002 09260 3
DUDLEY KNOX LIBRARY