AN INHERENTLY CONCURRENT LANGUAGE : A TRANSLATOR

by

CHUNMEI LIOU SIANG

B.S. Chung-Yuan College of Science and Engineering, 1972

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1980

Approved by :

Elizabeth Unger

ACKNOWLEDGEMENTS

I wish to express appreciation to my major advisor, Dr. Elizabeth Unger, for innumerable ways she provided ideas, assistance, and patience throughout my research project.

Much appreciation is due Shounan Wang, my friend. His intellect and enthusiasm were valueable in the completion of this work. As a mentor and friend he is a rare and inspirational example.

The support and encouragement of several individual made the pursuit of this degree a possibility. Included are Richard McBride, and Kam Mok.

Thanks are due also to the committee members, Dr. David Gustafson, and Dr. Rod Bates. I appreciate the time they took out of their busy schedules to serve on this committee.

Finally, specially gratitude is given to my entire family for the sacrifices they all have made during this venture. My husband, provided support in difficult times, good advice often needed to overcome doubts, and personal sacrifices made possible the completion of this report.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS.....ii
TABLE OF CONTENTS.....iii

CHAPTER 1: INTRODUCTION.....1
1.1: THE PROBLEM.....1
1.2: AVAILABLE TECHNIQUES.....3
1.3: PAPER CONTENTS.....9

CHAPTER 2: SOLUTION.....10
2.1: INTRODUCTION.....10
2.2: THE MODEL.....12
2.3: A LANGUAGE.....18
2.4: TARGET LANGUAGE.....21
2.5: DESIGN OF TRANSLATOR.....24

CHAPTER 3: ANALYSIS.....27
3.1: INTRODUCTION.....27
3.2: SAMPLE TRANSLATIONS.....32

CHAPTER 4: CONCLUSIONS.....38
4.1: RESULTS.....38
4.2: FUTURE WORK.....40

SELECTED BIBLIOGRAPHY.....41

APPENDIX A: ACM-L2 IN BNF

APPENDIX B: ACM-L2 IN LL1 GRAMMAR

APPENDIX C: TARGET LANGUAGE IN BNF

APPENDIX D: THE TABLES' FORMAT

APPENDIX E: TRANSLATOR

APPENDIX F: SAMPLE RUNS

APPENDIX G: THE LISTING OF ERROR MESSAGES

CHAPTER 1
INTRODUCTION

1.1 The Problem

In most existing programming languages, programs are executed sequentially, and the written order of the code determines the order of execution of program statements. This complied with the fact only one processor was available at any one time in the past. Thus Yesterday's machine capabilities have limited us to sequential processing in our programming languages.

Some programming languages which allow concurrent execution require programmers to create synchronization codes (e.g. wait and event in PL/1), or support the synchronization (monitor in CPASCAL, i.e., Concurrent Pascal) as the user specifies. In both cases, the concurrent capabilities are built upon the basic sequential processing scheme, and the user is required to understand the problem of synchronization. It is felt that this is not knowledge easily applied by most production programmers.

The evolution of microprocessors makes the multiprocessor architecture more promising. The computer programmer has been given machine capability to carry out more than one action at a time. A language that is inherently concurrent can more fully utilize the

capabilities of a multiprocessor machine. The model for a class of such languages has been developed by Unger (1978). The model named ACM model.

In the ACM model, several sections of a program that have no sequential relationship to one another are recognized for execution simultaneously (concurrently). The operations which are ready for execution are based on their input (material) availability, i.e., data driven, and other programmer specified conditions. Basically, the language model has only one statement format which has five parts: stimulation, action, materials (inputs), results (outputs), and termination. Stimulation is an optional condition which may be used by the programmer to augment the data driven sequencing of the execution of an operation (i.e., action). Termination is an optional condition which can be used by a programmer to terminate execution of an action. An action can be an operator or something similar to the traditional subroutine call. Materials and results have the more common meaning of inputs and outputs. Since this project uses the ACM model as the basis for its research, several concepts and definitions from the model will briefly be described in the next Chapter.

1.2 Available Techniques

The concept of data flow refers to the idea of a data-driven computation in which operations are initiated when all their input operands are available. This principle serves as a basis for Unger's ACM model. The idea was derived from the development of consequent procedures (Fitzwater and Schweppe, 1964) which allows program to proceed execution when required input data becomes available. "First Version of a Data Flow Procedure Language" (Dennis, 1974) describes a data flow procedure language which represented algorithms in a graphical manner similar to that used by Petri Nets. The paper by Dennis asserted that a compiler can ascertain the operations (actions) which are ready for execution, dependent only on their inputs' (materials) availability. The more developed forms of the data flow languages provide recursion and repetition. For this project, data flow models provide two basic ideas for the model language, namely, concurrency and a detection of ready-to-execute instructions.

Closely related to the data flow concept is that of single assignment (Comte, 1976a) under which a variable may be defined (assigned) with one unchangeable value during a given program execution. With this concept one will always obtain a deterministic result. LAU (Syre, 1977) is the only known partially-implemented language predicated upon intrinsic concurrency. It uses the single assignment

principle for most data objects. The deficiency of this approach is that the user has to define a different name for each assignment in order to simulate the more traditional notion of how a variable is used. However, single assignment can be violated in the implementation of LAU when the value of a data object is changed through the application of primitive synchronization constructs which are available in the language.

Consequent procedure networks (Fitzwater and Schweppe, 1964) which viewed programs as procedures (tasks) that can produce subprocedures to perform part of its functions. Each subprocedure can in turn produce subprocedures of its own, and so on until a subprocedure is completely defined. This technique is used in contemporary operating systems, and is most suitable in a multiprocessor environment.

Control constructs within conventional languages which are supported by structured programming techniques include subprocedures, alternation, caseation, and definite and indefinite loops. In this study's model, research has used the concept of conditional computation pioneered by Fitzwater and Schweppe to effect alternation. For example, guarded commands (Dijkstra, 1975), (Hoare, 1978) are used as the building blocks for repetitive and alternative constructs. In the alternation ("if...fi") construct of Dijkstra is one construct in which only one statement in the

construst is selected for execution and that statement must have a guard (condition) which has a "true" value. alternation ("if...fi") construct of Dijkstra in which but one statement must have a guard (condition) which has a "true" value. Caseation is considered by Unger (1978) to be a generalization of the alternation construct since it expands a choice between two computational paths into a choice among several paths. A repetitive construct supplies the basic loop mechanism. The repetitive ("do...od") construct of Dijkstra (1975) is one in which a statement is selected for execution if the guard value is true, and statement selection will continue until all guards have a value "false".

To be intrinsically concurrent, a language should be dependent only upon conditions which drive a program's actions. From the above basic concepts Unger (1978) developed and proposed the ACM model. The model expresses actions in linear syntax and is as similar to commonly accepted grammars as the following design attributes permit. The model:

- represents concurrent computations intrinsically,
- uses one encompassing information structure,
- supports conditional computation on any basis,
- provides alternate computation based on context,
- allows partial computations on incomplete data,

- expresses common computation in natural form, and
- encourages the structured development of programs.

Usually, programming statements consist of nouns and verbs. The nouns of this model are the names of its objects, while its verbs are the names of requests which describe an action. Generally, each object has an unchanging value, however, various incarnations of an object may exist over time, and each incarnation has its unique designator.

The actions of this ACM model are always described through effects upon objects. Requests which describe an action, by supplying material, cause the action to be effected and produce results when the action is complete. The action communicates with the invoking environment through a list of materials (inputs) and a list of results (outputs). All objects are local except for some globally declared variables . The sequencing of requests is accomplished through the availability of materials and the use of stimulations and terminations. The model is capable of naturally expressing various types of computation. The language derived from the ACM model has the ability to express all effective procedures and is capable of expressing nondeterministic computations.

The simulator used to accept the output of the translator is based upon a prototype CPASCAL program

referred to as NETSIM (Wallentine, 1978). NETSIM provides a simulation facility that is written in Concurrent Pascal and supports many of the same services that have been adopted by SOL (Knuth, 1964). The main body of NETSIM is coded as a monitor, SIMMON, which has entry points corresponding to the Sol-like services that it provides. In addition, this monitor may be modified to include a user's code so that it provides user defined services and statistics. Some of the basic services managed by this monitor include:

- the maintenance of a system clock,
- events which are SIGNALed and AWAITed to provide synchronization among the processes constituting the simulation scenarios,
- facilities that represent resources global to the processes,
- the delay and restarting of processes waiting for a logical predicate, phrased in terms of variables accessible to the monitor, to hold true.

The analogue of a transaction in SOL is represented by a user-supplied scenario written as a program in terms of Sequential Pascal statement and calls to the entry points of SIMMON. Portability of the system is assured since it has been written in an essentially unmodified version of Pascal (Neal, 1978). Further, the system is adaptable since it allows a user scenario to be modified by merely individual recompiling the desired scenarios rather than recompiling all of the scenario programs and the concurrent program containing SIMMON.

One of the user scenarios is designated as a MASTER-TIMER. This program has the responsibility for declaring the number of facilities and events that are available from SIMMON. In addition, it sets the duration for which the simulation is to run, and requests any desired statistics.

1.3 Paper Contents

This report is divided into four chapters. The problem and related concepts have already been described in Chapter 1. Chapter 2 provides the problem solution and includes the description of the concurrent model, the definition of the language mapped from the concurrent model, and presents the design of translator. In Chapter 3 an analysis of the translator along with some sample translations is presented. Finally, Chapter 4 summarizes the results of this project and describes possible directions for future research.

CHAPTER 2
SOLUTION

2.1 Introduction

In Chapter 1, we have described the problem of most existing programming languages, namely the lack of a natural expressive capability for programmer use. The solution of this problem was presented in the form of a model, for a class of inherently concurrent languages, which has been developed by Unger (1978). The ultimate purpose of this research is to study the underlying behavior of the concurrent model, ACM, through simulation of programs written in a language which is mapped from the ACM model. The results of these simulations can be used to determine where improvements need to be made.

The first step of this research project precisely defines the project language in Backus-Naur Form (BNF) which is then converted to an LL1 grammar. Next, a translator was written to transform the source program into a form close to the concurrent model and acceptable as the input to a simulator which be written by another graduate student R. McBride. The output of the simulator is used to analyze the behavior of programs in the language. The overall structure of the project is shown in Figure 1. In the next section, we shall briefly describe several concepts and definitions from the model.

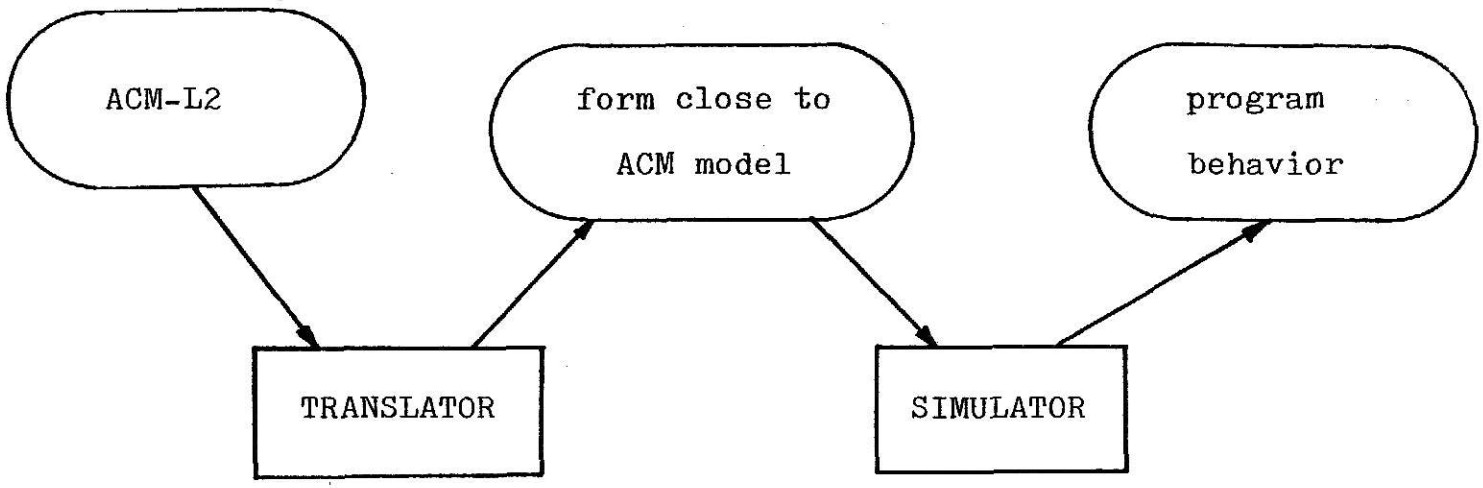


Fig. 1 Overall structure of the project
program behavior used to study the property of the ACM model

2.2 The Model

Except for the declaration statement, all statements of the ACM model are in one format, namely SMART. The first component, S, represents the stimulation condition which is a simple or complex conditional expression that can be reduced to a Boolean value. The operation, A, cannot be executed unless the value of stimulation condition is true. This allows the programmer further control before the start of execution. M, the material, is a list of necessary input objects. A is the name of an action. R, the results, is a list of necessary output objects. T, which is the termination condition, is a conditional expression as was S. The action A should be terminated whenever the value of the termination condition becomes true. This provides the programmer with control of the executions of the actions. An attempted request is either never started, aborted, or completed normally. Any of five component may be null except for the action name.

The universe of possible objects, O, in the model is the cartesian product of five component spaces: O=D*A*R*C*V. Thus, an individual object is defined as a five tuple of components o=(d,a,r,c,v). $d \epsilon D$ is the designator which is a complex of names that identifies the object and includes a user-defined name (u), the context of creation, and identifies the particular instance (which may be spatial or chronological). $a \epsilon A$ is the attribute, and it contains

information about the underlying atomic representation, the internal structure, and the external relationships of the object. $r \in R$ is the representation, and it deals with the characteristics of the physical storage of the object. $c \in C$ is the corporality which provides information about the longevity of the object's existence, the environment in which the object exists, the number of replications and their availability, and the authorization for use of the object. $v \in V$ is the object's value, and it is either an atomic value, a number of atomic values, an object, or a number of objects compatible with the same attributes.

The longevity or length of existence of objects in a process is a continuous spectrum. Within the ACM model, this continuous spectrum of longevities has been broken into four categories: fixed, static, dynamic, and fluid, i.e., the longevity subcomponent of the corporality component of an object must have one of four values: fixed, static, dynamic, or fluid. By default, objects will be considered as dynamic. These four values are ordered above in terms of ability to allow change.

Objects with a fixed longevity are those objects for which, in a wide frame of reference, the components may not change. Fixed objects exist prior to the existence of the context in which they are used.

Definition 2.2-1. Objects with longevity fixed are defined during the inception of the model. The value of this type of object is bound to the other object components during the existence of the model.

Static longevity is similar to fixed longevity, except that a smaller environment exists within which each of the object components is bound to the object. Once each of the object's components comes into existence, it is bound to the object in all environments. As such, a static constant would not be created prior to the model but would be created within the system.

Definition 2.2-2. Objects with longevity static are defined by actions within the model. Once a component of this type of object exists, it is bound to the object.

A longevity of dynamic is used to describe objects whose value may change over time and each value change causes a new distinguished incarnation to exist.

Each time the value is created, the chronological identity is changed to the new reading of its clock, i.e., a new sequence number is generated, thus creating a new object. Within the model previous incarnations of objects could be purged by a specific action. In an implementation, such specific actions or preferably a management information scheme would be required.

Definition 2.2-3. Objects with longevity dynamic are

> defined by actions within the model and may appear to change value to the user using just the u part of the designator. Such an object must have a chronological identity, t, and/or a sequence number, 0. Each new value associated with the object creates a new incarnation with an increased sequence number and/or changed chronological identity. An unqualified reference to such an object retrieves the most recent incarnation. Relative references d.-n, n>=0 retrieves the incarnation created n steps previouly. When n=0 the current incarnation is specified.

The final category for longevity is fluid which indicates that the object value may change, but subject to the restriction that change must occur between uses. Also, only the current value of this type of object is maintained.

> Definition 2.2-4. Objects with longevity fluid are defined by actions within the model. They may change value at any time between uses. No previous incarnations are available.

An action is an object whose value consists of requests for more detail actions. A request is a statement indicating an action to be performed which is analogous to the call of the subroutine in traditional languages or an assignment statement. The form of the request is defined as follows:

> <request>::=(s,m,a,r,t)

where s,m,a,r, and t represent the components of the five tuple defined before.

An action may be detailed or given computational particulars by expressing it as a set of one or more requests. Specification of the detail of an action analogous to a subroutine definition in conventional languages is achieved with a header tuple $(s_o, m_o, a_o, r_o, t_o)$ followed by a box containing the requests. An example is provided in Figure 2.

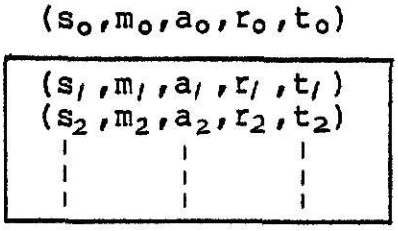


Figure 2. An action and its details

The conditions which appear on the header tuple are referred to as internal conditions while those conditions which appear on a request are referred to as external conditions. Both internal and external stimulation conditions serve as a guard to the execution. The internal stimulation condition provides the ability to the programmer to indicate when an action cannot be executed. The external stimulation conditions allow the user to control his particular problem solution. The internal termination condition is tested before the start of the requests of a detailing action and at the completion of all requests. If the termination condition is false, the action will be executed continuously. For this reason, the internal

termination condition serves as the looping mechanism. Thus, definite and indefinite loops are provided in the model. The external termination condition provides the user with the ability to terminate its execution at any time with no results returned.

The ordering of execution among the requests within the model is provided by data driven. Data driven is the principle for the ordering of execution among simple requests. A request is eligible to execute if

- the action is available,
- each requisite data object in the material list is complete and available, and
- the user is authorized to use each object involved.

In other words, the order of appearence of requests having a common local context has no influence upon the order of execution. The sequencing of two requests can be determined by the problem solver through the appropriate selection of material and result object names. Further control of the order of execution is available through the use of conditions as previously discussed.

2.3 The language

Using the different subsets of the ACM model, we can construct different languages. For convenience, the language we will consider is called ACM-L2. The definition of ACM-L2 is derived from one-to-one mapping from the ACM model and is expressed in BNF. The BNF description of ACM-L2 which is presented in Appendix A is derived from a basic one-to-one mapping from the subset of the model defined above. We mentioned before, an action is an object whose value consists of requests for more detail actions. A request is a statement indicating an action to be performed which is analogous to the call of the subroutine in traditional languages or an assignment. The form of the request is defined in the language as follows:

<request>::=[<s>]<a>(<r>:<m>)[<t>];

where s,m,a,r, and t represent the components of the five tuple defined in the previous section.

Several examples of such requests are:

```
PROC LOOP(A.+1,B.+1:A.+0,B.+0,N)[INC I(3,+1,9)];
:=(X,Y,Z:1);
+(A.+1:A.+0,B.+0);
[N<ANO]-(R:N,1);
[N>=ANO]REST(R:N);
```

The data objects are simplified in that all object names are assumed to have an implicit chronological name, and no construct is provided for user to control or to access the information provided by the representation or corporality attributes of an object.

As well as omitting some constructs provided in the model, certain clarifications and extensions to the model are made. The first change is really an extension of the model to provide a compact notation for the iterative DO. This type of notation is available in most common programming languages and fits quite naturally into the internal termination of the detail of an action. The construct defined allows a programmer to set up a loop counter by specifying a variable, its initial value, the value of its increment, and its final value.

The second clarification deals with time names for variables. The model allows use of relative time names on variables (e.g., X.+1,X.+0,X.-1). For this research it is assumed that the relative time names were relative to the specific execution of the model, so the X.+0 or X refers to the current value of X when the action was invoked. The clarification eliminates the need for often clumsy ordering of statements when a specific instance of a variable is required (i.e., a sequence of statements such as X=X+Y, A=X*B can be clarified as +(X.+1:X.+0,Y), *(A:X.+1,B)).

The model allows multiple assignments of two types. The first type occurs when several output objects receive the same value. For example, :=(X,Y,Z:1) where X,Y,and Z all simultaneously receive the same value of one. This is an indivisible operation. The second type occurs when several assignments could be executed concurrently and can thus be combined as one action. For example, :=(X,Y,Z:1,2,3) would make the assignments X:=1, Y:=2, and Z:=3. The model provides for an action on an object, thus concurrent assignment to all or part of an array can occur.

The third clarification deals with conditional expressions. In the stimulation and termination conditions, complex Boolean and arithmetric expression are allowed. In order for an action to begin or end because of one of these conditions, the value of each subcondition must be considered at the same time. For example, if there are several conditions connected with "ANDs", then all must be true at the same instance for the action to be correctly invoked.

2.4 Target Language

The target language is transfered from a one-to-one mapping of the source language (ACM-L2) and is expressed in BNF. The BNF description of the target language presented in Appendix C.

All statements of target language are the same as the statements of source language and have five components: S,M,A,R , and T. For convenience of implementation, certain of the SMART components are converted to pointers or indices into tables. The delimiter "$" is used to seperate the five components, while ";" is used at the end of SMART tuple (statement), and "," is used as the list delimiter. The output of translator are the target program, which is stored in the procedure table, and certain tables of information. These tables include: the stimulation table (which stores stimulation conditions), the termination table (which stores termination conditions), the dictionary table (containing information about variables and procedures), and the fix table (that stores the constants of the program). There are some tables actually built upon at execution time: the static table (which stores the values of the static variables), the fluid table (that keeps track of the values of fluid variables) , and the dynamic table (which stores the values of dynamic variables).

The first component of one statement in the target

language is an index of stimulation table,the second and fourth components are the lists of indices of dictionary table entries. If data objects of the statement in the source language are constants then they will appear as lists of indices of the fix table entries. The data objects types are independent of one another and the lists will reflect the occurrence of types. The fifth component is an index into the termination table. If the action name of the statement in the source language is a procedure name then the third component should be the index of dictionary table, otherwise it will be the actual characters as they appear in the source language.

For example, consider the following statement.

$SINDEX$XQ,YQ$+$RQ$TINDEX; where

SINDEX: is an index into the stimulation table which points to the action's stimulation condition,

XQ,YQ : are the indices into the dictionary or the fixed table which point to the input objects,

RQ : is an index into the dictionary table which points to the output object,

+ : is the arithmetric operator (action name),

TINDEX: is an index into the termination table which points to the action's termination condition.

Any of the five components may be null except the action name. All of the SMART components are mapped from the source language. Two additional delimiters have been introduced, the end of one procedure is marked "!", while

the beginning and the end of the program are marked by "."

2.5 Design of Translator

The tasks of the translator are to perform the following actions for each line of source code.

- Syntactic action. The scanning of the source lines, as received from the input/output module, is performed in order to check its syntax and report any errors back to the user.
- Semantic action. The translating of the source line into its target language equivalent, along with checking its semantics and reporting any errors back to the user. As a source line is translated, its target language equivalent is built up along with information tables for incorporation into target program.

There are two important principles affecting the translator's actions. First, each source statement except declaration statement is translated as an entity in itself, independent of all the other source statements. Second, a statement which has been found to contain a syntax error or semantic error still is included in the target language program.

The source language is designed to be scanned from left to right and it intersperses semantic actions with syntactic actions. In other words, it is a one-pass translator that acts as a parser with semantic actions hung on in the appropriate places. In this case the target program is gradually built up as the scan proceeds along the source line.

The diagram of the translator's design is shown in Figure 3 . Every time the parser wants to advance its scan, it calls the subroutine GET_TOK. Different classes of tokens will search different tables to check whether a token exists or not. If it exists, the token's class and index of the table in which it located is returned. Otherwise a new table entry is created for the token, and then the class and an index indicating this entry is returned.

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

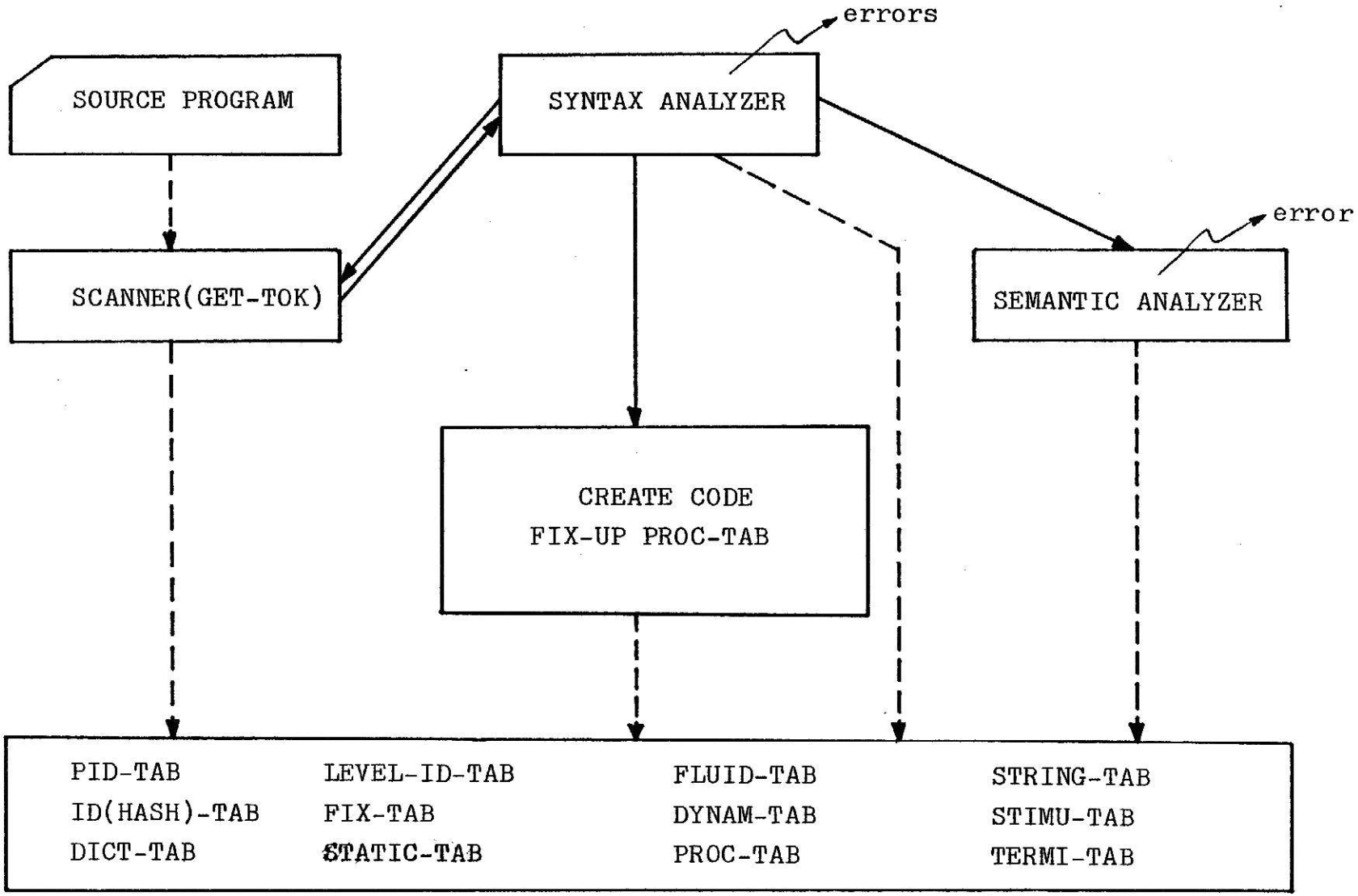


Fig. 3 Design of translator

CHAPTER 3
ANALYSIS

3.1 Introduction

There are certain circumstances which cause designers to favour a multi-pass translator. One special reason for a multi-pass translator is that the machine on which the translator runs is relatively small. If a one-pass translator would occupy 10000 instructions and the machine can hold only 2000 instructions, then clearly some splitting up is necessary. The second reason for a multi-pass translator is to make it perform better in a virtual storage environment. In such an environment, it is better to have small sections of code which are heavily used and then discarded, as each pass of a multi-pass translator might be, rather than a larger section of code, such as a one-pass translator, where each instruction is used comparatively less frequently. Finally, a translator is split into passes because it is so complicated that nobody can understand it properly until it is packaged into smaller tasks.

In this project, the source and the target language are very simple. Both languages employ one type of format statement, as has been described in Chapter 2. Therefore, none of the reasons for making the translator multi-pass holds, and that semantic actions are interspersed with syntactic ones. Thus, in this situation a one-pass translator can be successfully used. A danger with a

one-pass translator is that it becomes incomprehensible to everyone except its author, and eventually incomprehensible even to him. It is therefore important to divide the translator into logically separate modular units. In a one-pass translator, the interactions between modules are stronger as they all work together to process each source line. As an aid in understanding the translator, we will use pseudo_code to describe the translator's main and most complicated part.

We would like to describe some tables which along with target program are the input of the simulator. The format of the tables is shown in Appendix D.

DICT_TAB : stores the names of all identifiers, procedures, variables, and information related to them.

FIX_TAB : stores the constants which may be real, integer, string, or Boolean.

STIMU_TAB : stores the character strings corresponding to stimulation conditions which appear in source program.

TERMI_TAB : stores the character strings corresponding to termination conditions which appear in source program.

PROC_TAB : stores the target program's statements and any user defined procedures.

There are two tables, ID_HASH_TAB and PID_TAB that help to build up DICT_TAB. ID_HASH_TAB stores the identifiers with the exception of procedure names, while PID_TAB stores the all procedure names. In this case, one

procedure can have the same name as another identifier as long as it is not a procedure name.

The statement of the target language is in one format, SMART, in which every statement includes five components:

1. stimulation condition,
2. material list,
3. action name,
4. result list,
5. termination condition.

There are five special variables to store this information.

1. CSNO : stores the index of the current stimulation in the STIMU_TAB.
2. CMNO : stores the input material list, which covers the FIX_TAB's index and the DICT_TAB's index.
3. CANO : stores the operator or the index of the current procedure name in the DICT_TAB.
4. CRNO : stores the output result list which covers the DICT_TAB's index.
5. CTNO : stores the index of the current termination in the TERMI_TAB.

The contents of the above variables are characters.

Now, let us use pseudo_code to describe the translator's main part.

```
PROCEDURE REQUESTS;
  BEGIN
    REPEAT
      IF(AUNIT.CLAS='[       ')
        THEN CALL STIMU AND RETURN CSNO;
      BUILD UP CANO;
      "IF THERE IS NO ACTION NAME, CREATE ERROR
       MESSAGE"
      IF(AUNIT.CLAS='(       ')
        THEN CALL PARM_ARG AND RETURN CRNO,CMNO;
      IF(AUNIT.CLAS='[       ')
        THEN CALL TERMI AND RETURN CTNO;
      CALL CREATE_CODE;
      "COLLECTS THE ABOVE FIVE PARTS AND CREATES ONE
       SMART TARGET STATEMENT"
      CALL UP_CODETAB;
      "INSERTS THE SMART STATEMENT INTO PROC_TAB WHICH
       STORES THE TARGET PROGRAM "
      CURRENT_ST := SUCC(CURRENT_ST)
    UNTIL(AUNIT.CLAS='END    ')
  END;
```

The previous routine handles the all requests in one source procedure.

```
PROCEDURE
SAME_PROC(AAPID:INTEGER;ATI,ATC:THREE_ARY);FORWARD;
PROCEDURE ACTIONDCL;
  BEGIN
    WHILE(AUNIT.CLAS='PROC   ') DO BEGIN
      GET_TOK;
      IF(AUNIT.CLAS='[       ')
        THEN CALL STIMU AND RETURN CSNO;
      BUILD UP CANO;
      CURRENT_LEVEL:=SUCC(CURRENT_LEVEL);
      IF(AUNIT.CLAS='(       ')
        THEN CALL PARM_ARG AND RETURN CRNO,CMNO
        ELSE CREATES ERROR MESSAGE;
      IF(AUNIT.CLAS='[       ')
        THEN CALL TERMI AND RETURN CTNO;
      CALL SAME_PROC(APID,AATI,AATC);
      "AT THE END OF THE PROCEDURE MARK "!" IN PROC_TAB
       AND FIX THE PROCEDURE'S LENGTH "
    END "WHILE"
  END;
```

The preceeding routine handles the all procedures which defined in one procedure.

```
PROCEDURE SAME_PROC;
  BEGIN
    CURRENT_ST:=SUCC(CURRENT_ST);
       CALL CREATE_CODE;
       CALL UP_CODETAB END;
    IF(AUNIT.CLAS='CONT    ') THEN CALL CONTDL;
    "HANDLES CONSTANT DECLARATION AND STORE THESE
     INFORMATION TO THE FIX_TAB AND DICT_TAB"
    IF(AUNIT.CLAS='VAR     ') THEN CALL VARDCL;
    "HANDLES VARIABLES DECLARATION AND STORES THESE
     INFORMATION TO THE DICT_TAB "
    CALL ACTIONDCL; "TO DEFINE PROCEDURES"
    CALL REQUESTS;  "TO HANDLE THE REQUESTS"
    CALL ARRANGE_ID_TAB;
    "TO DELETE THE CURRENT_LEVEL'S ID. WHEN IT GO THROUGH
     THIS LEVEL "
    CURRENT_LEVEL:=CURRENT_LEVEL-1;
    IF(AUNIT.CLAS='END     ')
      THEN GET_TOK ELSE CREATES ERROR MESSAGE;
    IF(AUNIT.CLAS='PID     ') THEN GET_TOK
  END;
```

The previous routine handles the whole block except the first statement.

```
PROGRAM MAIN;
  BEGIN
    CALL INITIALIZE;
    CALL MAIN_HEAD;
    "INSERTS "." TO PROC_TAB TO MARK THE BEGINNING OF
     PROGRAM"
    CALL SAME_PROC;
    CALL MAIN_TAIL
    "INSERTS "." TO PROC_TAB TO MARK THE END OF THE
     PROGRAM"
  END.
```

In this translator, the maximum length of the identifier is 7. If there exists an identifier with a length greater than 7, then the translator will send message "name too long" to the user. The maximum length of a string is defined 10. If a string's length is greater than 10, then the user will get a message "illegal string". See Appendix G for other messages which may occur.

3.2 Sample Translations

In this section, we will discuss some sample translations. The first topic we will talk about is declaration statements. Although the target language program did not cover declaration statements, instead there are some tables (namely, FIX_TAB, DICT_TAB, STIMU_TAB, TERMI_TAB, and PROC_TAB) which are output by the translator along with the target program as input to the simulator. So, when a declaration statement is encountered, relevant information is stored to the appropriate table by the translator.

For example, consider the following sequence of declarations:

```
CONT ANO=10;
VAR  A,B,C,D : INTEGER,DYNAMIC;
```

The translator will check whether ANO,A,B,C, and D already exist in ID_HASH_TAB . If one of them existed there, it is a redefined variable, otherwise it will create an entry in DICT_TAB. The default attribute of a variable is assumed to be real, and the default longevity is assumed to be dynamic. The above statements create entries in the DICT_TAB which is shown in Figure 4.

USER	CONTEXT	NEXT_SEQ_NO	ATTRIBUTE	LONGEVITY	FIELDS USED ONLY AT RUN TIME	VALUE_PTR
.	.	.	.	.		.
.	.	.	.	.		.
.	.	.	.	.		.
ANO	*	0	1	1		1
A	*	0	1	3		4
B	*	0	1	3		3
C	*	0	1	3		2
D	*	0	1	3		1
.	.	.	.	.		.
.	.	.	.	.		.
.	.	.	.	.		.

* CURRENT CONTEXT

Figure 4. DICT_TAB

The fields of DICT_TAB are described in the following manner.

USER: this field stores a simple name of 7 characters which created by the user.

CONTEXT: stores the context of creation for the object that is represented by a finite ordered sequence of zero or more names, these names are represented by integers. Context names are generated when an object passes from one context to another in a hierarchy of contexts.

NEXT_SEQ_NO: only objects with longevity dynamic need to use this field, because an object with longevity dynamic must have a chronological identity, or a sequence number. Each new value associated with the object creates a new incarnation with an increased sequence number or changed chronological identity.

ATTRIBUTE: stores the type of object's value, which can be either integer, real, string, or Boolean. We used 1 to represent integer, 2 to represent real, 3 to represent strings, 4 to represent Boolean, and 0 to represent all others.

LONGEVITY: stores the object's longevity, which is either fixed, static, dynamic, fluid, or procedure. We used 1 to represent fixed, 2 to represent static, 3 to represent dynamic, 4 to represent fluid, 5 to represent proc, and 0 to represent all others.

VALUE_PTR: stores an index into the FIX_TAB, the STATIC_TAB, the DYNAM_TAB, the FLUID_TAB, or the PROC_TAB which points to the object's value.

The second topic that we discuss in detail concerns the handling of the first statement of a procedure's definition. Let us consider the following statement.

```
PROC LOOP(RSLT.+1,LTRS.+1:RSLT.+0,LTRS.+0,N)
     [INC I(3,+1,9)];.
```

In addition to the three materials which supply input to the procedure LOOP, there is also an implicit, system-defined

parameter which is supplied by the translator. This parameter, which is invisible to the user, gives the number of statements which make up the procedure body in the target language. The stimulation does not exist in this statement, so that CSNO=' '. The action name LOOP, which is the procedure's name, is assumed to have an index in DICT_TAB of 7, and so CANO='7 '. The input elements may be constants or variables which will be translated as indices into FIX_TAB or DICT_TAB. We use "F" and "D" to distinguish these two tables. If we assume that DICT_TAB indices of RSLT,LTRS, and N are 8,9, and 10, then CMNO='D8.+0,D9.+0,D10 '. All output elements are translated to DICT_TAB's indices, So we do not need to use "F" or "D" to mark them, so CRNO='8.+1,9.+1 '.

The termination condition is [INC I(3,+1,9)], and we assume the DICT_TAB index for I is 11 and that the FIX_TAB's indices for the constants 3, 1, and 9 are 1, 2, and 3, respectively. The routine TERMI transforms the condition [INC I(3,+1,9)] into the string 'FOR D(11):=F(1) TO F(2)BY+F(3)DO' and then stores it into TERMI_TAB. The T component of the procedure definition is replaced by this string's index in the TERMI_TAB. We assume this index is 1, so that CTNO='1 '. When the delimiter ";" is encountered, we collect above five parts to one SMART statement as follow: $$D8.+0,D9.+0,D10$7$8.+1,9.+1$1;.

In this statement, the parameters and the termination condition's index I were not defined. Type checking can not be performed until the procedure's declarations are finished. The stimulation condition belongs to the calling environment. So that the stimulation's operands are not defined in the environment of the called procedure.

The third topic that we will discuss concerns requests. Every request consists of five components: S, M, A, R, and T. All the components may be null except action name (i.e., A). If there is no stimulation condition then CSNO=' '. However, if it does exist, then the routine STIMU will analyze it and return the starting position of the stimulation string in STIMU_TAB. For the action, if it is a procedure name then the value of CANO will be the position of this action name in the DICT_TAB. If it is an operator, then CANO will store the actual char. When there is no termination condition then CTNO=' ', otherwise the routine TERMI will arrange it and return the position of the termination in TERMI_TAB. In constructing the translator, any Boolean expression was allowed to be a stimulation condition, while termination conditions were restricted to being a form of DO loop. When the delimiter ";" is encountered these five parts are collected into one SMART statement.

An example of such a translation is:

source statement	target statement
+(RSLT.+1:RSLT.+0,LTRS.+0);	$$D8.+0,D9.+0$+$8$;
:=(LTRS.+1:LTRS.+0);	$$D8.+0$:=$9.+1$;
[N<ANO] -(RSLT:N,1);	1D6,F3$-$4$;
[N>=ANO] REST(RESULT:N);	11D6$12$4$;

The routine STIMU will transform [N<ANO] into "D(6)<D(2);", and store this string into the STIMU_TAB, and finally return the starting position of the string in the STIMU_TAB. Similarly, [N>=ANO] will be changed into the string "D(6)>=D(2)".

Finally, we will discuss the end of every procedure when "END proc_name;" is encountered. Beside performing a syntax check, the translator also checks the procedure name to ensure that the correct procedure definition is terminated, and adds "!" to the PROC_TAB. But, at the end of the main program "." is added to the PROC_TAB.

CHAPTER 4
CONCLUSIONS

4.1 Results

We mentioned before that the ultimate purpose of this research is to study the underlying behavior of the ACM model and then to determine where improvements can be made. This research constructed a translator that takes ACM-L2 to a form suitable as input to a simulator for concurrent processing. The output of the simulator is to be used to demonstrate the behavior of program in the language. The major results of this master's research as follows:

1. The definition in BNF of a source language mapped from the ACM model. This language is similar to one defined and used in the research done by Karla Marietta in 1979 (see Appendix A).

2. The transformation of the BNF into an LL1 grammar for which it is convenient to write translator (see Appendix B).

3. The definition in BNF of the target language, acceptable as the input to a simulator to be written by R. McBride (see Appendix C).

4. The construction of a translator and associated information tables along with the target language as the input to the simulator. The format of the tables show in Appendix D.

5. The translator (see Appendix E) and the sample runs of a translator to accomplish the source language to target language translation (see Appendix F).

The mapping of the language ACM-L2 into a Pascal based system demonstrates that the advantages of the ACM model can be realized in a usable programming language. It achieves the goals of facilitating utilization of concurrent

capability and encouraging structured programming. The availability of chronological identifiers provides an environment that closely conforms to actual information needs by allowing both absolute time and relative time as qualification to the identity of an object. Further study of ACM-L2 will help determine if languages mapped from ACM model are worthy of consideration for the first generation of intrinsically concurrent languages.

4.2 Future Work

We have supplied a definition and a translator for the language ACM-L2. Topics for future research center upon issues of syntax and semantic designs. Based on this study, six major areas for further research are proposed as follows:

1. write the simulation program,
2. run the simulation program,
3. analyze the results,
4. expand the semantics to allow recursion,
5. expand the concept of data type, (e.g., in this research arrays and record types were never considered),
6. expand the constructs (e.g., indefinite loops).

After expanding these areas, the resulting language would be more powerful and varied.

SELECTED BIBLIOGRAPHY

[Bohm 66] Bohm, C., and Jacopini, G. "Flow Diagrams, Turing Machine, and Languages with only two Formation Rules",CACM vol. 9-5, May 1966, pp. 366-371.

[Brown 79] Brown, P. J. Writing Interactive Compilers and Interpreters, New York : Wiley-Interscience, 1979.

[Chamberlin 71] Chamberlin, D. D. "The Single Assignment Approach to Parallel Processing", IBM Research Report, RC 3308, 1971.

[Comte 76] Comte, D., and others. "Parallelism, Control and Synchronization Expression in a Single Assignment Language", Fourth annual ACM Computer Science Conference, Anaheim, California, 1976.

[Dennis 74] Dennis, T. B. "First Version of a Data-Flow Procedure Language", Project MAC Computational Structure Group Memo 109-1, 1974, MIT, Cambridge, Massachusette.

[Dijkstra 75] Dijkstra, E. W. "Guarded Commands, Nondeterminacy and Formal Derivation of Programs". CACM vol. 18-8, August 1975, pp. 453-459.

[Donovan 72] Donovan, J. J. Systems Programming, New York : McGraw-Hill,1972.

[Fitzwater 64] Fitzwater, D. R. , and Schweppe, E. J. "Consequent Procedures in Conventional Computers". Processings of the Fall Joint Computer Conference, vol. 28, pp. 465-476.

[Gries 73] Gries, D. Compiler Construction for Digital Computers, 1973.

[Hankley 79] Hankley, W. ,Wallentine, V. , Skidmore, A., and McBride, R. "NETSIM : Distributed Network Stimulation Program". Technical Report, CS79-02, Kansas State University, Manhattan, Kansas, January 1979.

[Hoare 78] Hoare, C. A. R. , "Communicating Sequential Processes". CACM vol. 21, no. 8, August 1978. pp. 666-677.

[Jensen 74] Jensen, K. , and Wirth, N. Pascal User Manul and Report, New York : Springer-Verlag, November 1974.

[Lewis 76] Lewis, P. M. , Rosenkrantz. D. J. , and Stearns, R. E. Compiler Design Theory, California : Addison-Wesley, May 1976.

[Peterson 77] Peterson, J. L. "Petri Nets". ACM Computing Surveys, vol. 9, no.3, September 1977, pp. 223-252.

[Unger 79] Unger, E. A. "A Concurrent Model : Basic Concepts". Proceedings of European Conference on Parallel and Distributed Processings, 1979.

[Unger 78] Unger, E. A. "A Natural Model for Concurrent Computation". Dissertation, University of Kansas, Lawrence, Kansas, 1978.

[Wallentine 78] Wallentine, V. , Hankley, W. , and McBride, R. "SIMMON-- A Concurrent Pascal Based Simulation System". Technical Report, TR79-05, Kansas State University, Manhattan, Kansas, April 1978.

[Williams 78] Williams, M. H. , and Ossher, H. L. "Conversion of Unstructured Flow Diagram to Structured Form". The Computer Journal, vol. May 1978.

APPENDIX A
SOURCE LANGUAGE ACM-L2 IN BNF

PROGRAM:

1 <program>::=program<progid>(<constant>);<contdcl>
<typedcl><vardcl><actiondcl>begin<requests>
end<progid>.

2 <progid>::=<identifier>

DECLARATION:

3 <contdcl>::=cont<contail>;<contdcl>|ϵ

4 <contail>::=<identifier>=<constant>;<contail>|ϵ

5 <vardcl>::=var<vartail>;<vardcl>|ϵ

6 <vartail>::=<idlist>:<basic attribute>,<longevity>
;<vartail>|ϵ

7 <idlist>::=<identifier><idtail>

8 <idtail>::=,<identifier><idtail>|ϵ

9 <longevity>::=fixed|static|dynamic|fluid|proc|ϵ

ATTRIBUTE:

10 <attribute>::=<basic attribute>|<new attribute>

11 <basic attribute>::=integer|real|boolean|char

12 <new attribute>::=<identifier>

ACTION:

13 <actiondcl>::=<action block><action dcl>|ϵ

14 <action block>::=proc<internal stimulation>
<action name>(<parmlist>)
<internal termination>
<contdcl><vardcl><action dcl>begin
<requests>end<action name>;

15 <action name>::=<identifier>

16 <internal stimulation>::=[<bool expr>]|ϵ

17 <internal termination>::=[<bool expr>]|ϵ

18 <parmlist>::=<result parms>:<material parms>

19 <result parms>::=<designator list>

20 <designator list>::=<designator><designatortail>|ϵ

21 <designatortail>::=,<designator><designatortail>|ϵ

22 <material parms>::=<designator list><constantlist>

23 <constantlist>::=,<constant><constantlist>|ϵ

REQUEST:

24 <requests>::=<request><arequests>|ϵ

25 <arequests>::=;<requests>|ϵ

26 <request>::=<external stimulation><request name>
(<arglist>)<external termination>

27 <request name>::=<action name>|<arith op>|:=

28 <external stimulation>::=[<bool expr>]|ϵ

29 <external termination>::=[<bool expr>]|ϵ

30 <arglist>::=<result args>:<material args>

31 <result args>::=<designator list>

32 <material args>::=<designator list><constantlist>

33 <constantlist>::=,<constant><constantlist>|ϵ

As well as omitting some constructs provided in the model, the first change is really an extension of the model to provide it with a compact notation for the iterative DO. The construct is defined to allow a programmer to set up a loop counter by specifying a variable, its initial value, the value of its increment and its final value. Actually, in this translator, we only considered this kind termination (see Appendix B: <termination condition>).

DESIGNATOR:

34 <designator>::=<user name><instance>
35 <user name>::=<identifier>
36 <instance>::=.<sequence no>|ϵ
37 <sequence no>::=<relative seq>|<absolute seq>
38 <relative seq>::=<sign><sno>
39 <sno>::=0|1
40 <absolute seq>::=<unsigned integer>

BOOL EXPR:

41 <bool expr>::=<expr>
42 <expr>::=<sexpr><exprtail>
43 <exprtail>::=<expr op><sexpr>|ϵ
44 <expr op>::=>=|=|<=|<>|>|<
45 <sexpr>::=<uop><term><sexprtail>
46 <uop>::=<sign>|ϵ
47 <sexprtail>::=<sexpr op><term><sexprtail>|ϵ
48 <sexpr op>::=+|-|or
49 <term>::=<factor><termtail>
50 <termtail>::=<term op><factor><termtail>|ϵ
51 <term op>::=*|/|and
52 <factor>::=<constant>|<identifier>|not<factor>|
(<expr>)|ABS(<sexpr>)|V_EXIST(<identifier>)

BASIC ELEMENT:

53 <identifier>::=<letter><lettertail>
54 <lettertail>::=<alphanum><lettertail>|ϵ
55 <alphanum>::=<letter>|<digit>
56 <letter>::A|B|C|D|...|X|Y|Z|a|b|c|...|x|y|z
57 <digit>::=0|1|2|3|3|5|6|8|9
58 <string>::=''|'<symbol><stringtail>'
59 <stringtail>::=<symbol><stringtail>|ϵ
60 <symbol>::=<letter>|<digit>|<special symbol>
61 <special symbol>::=any ASCII code
62 <constant>::=<integer>|<real>|<string>|true|false
63 <integer>::=<signed integer>|<unsigned integer>
64 <signed integer>::=<sign><unsigned integer>
65 <unsigned integer>::=<digit><digitail>
66 <digitail>::=<digit><digitail>|ϵ
67 <real>::=<digitail><fpart><epart>
68 <fpart>::=.<digitail>|ϵ
69 <epart>::=e<digitail>|ϵ
70 <sign>::=+|-
71 <index>::=<unsigned integer>

APPENDIX B
ACM-L2 IN LL1 GRAMMAR

17 <internal termination>::=[<bool expr>]|ϵ
18 <parmlist>::=<result parms>:<material parms>
19 <result parms>::=<designator list>
22 <material parms>::=<designator list><constantlist>

are converted as follows:

<internal termination>::=<termination condition>|ϵ
<parmlist>::=<result parms><material parms>
<result parms>::=<designator list>
<material parms>::=:<designator list><constantlist>

29 <external termination>::=[<bool expr>]|ϵ
30 <arglist>::=<result args>:<material args>
31 <result args>::=<designator list>
32 <material args>::=<designator list><constantlist>

are converted as follows:

<external termination>::=<termination condition>
<arglist>::=<result args><material args>
<result args>::=<designator list>
<material args>::=:<designator list><constantlist>

<termination condition>::=[<Inc expression>]
<Inc expression>::=Inc <user name>(<unsigned integer>
,<integer>,<fvalue>)
<fvalue>::=<user name>|<unsigned integer>

44 <expr op>::=>=|=|<=|<>|>|< are converted to

<expr op>::=><gtail>|=|<<ltail>
<gtail>::==|ϵ
<ltail>::==|>|ϵ

58 <string>::=''|'<symbol><stringtail>'
62 <constant>::=<integer>|<real>|<string>|true|false
67 <real>::=<digitail><fpart><epart>

are converted as follows:

<string>::='<stringer>
<stringer>::='|<symbol><stringtail>'
<constant>::=<real>|<string>|true|false
<real>::=<integer><fpart><epart>

APPENDIX C
TARGET LANGUAGE IN BNF

PROGRAM:

```
<program>::=.<blocks><requests>.
<blocks>::=<procedure block><blocks>|ϵ
```

PROCEDURE:

```
<procedureblock>::=$<sindex>$<mparms>$<proc_name>$<rparms>
                   $<tindex>;<requests>!
<sindex>::=<index>|ϵ   "stimu_tab's index"
<tindex>::=<index>|ϵ   "termi_tab's index"
<mparms>::=<pacrlist>
<rparms>::=<pacrlist>
<pacrlist>::=<pacr><listail>
<listail>::=,<pacr><listail>|ϵ
<pacr>::=D<index>,+<sno>|D<index>,-1|<index>,+<sno>|<index>|
         F<index>|<index>,-1|D<index>,<index>|<index>,<index>|
         %D<index>
<index>::=<digit><index>|ϵ
<sno>::=0|1
<proc_name>::=<index>   "dict_tab's index"
```

REQUESTS

```
<requests>::=<request><requests>|ϵ
<request>::=$<sindex>$<mparms>$<request_name>$<rparms>
           $<tindex>;
<request_name>::=<proc_name>|<arith_op>
<arith_op>::=+|-|*|/|:=
```

ILLEGIBLE DOCUMENT

THE FOLLOWING DOCUMENT(S) IS OF POOR LEGIBILITY IN THE ORIGINAL

THIS IS THE BEST COPY AVAILABLE

APPENDIX D
THE FORMAT OF TABLES

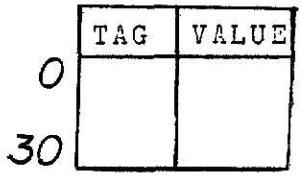
DICT_TAB:

	UR	CTXT	NSN	ATT	LON	AVL	COHO	COPY	VA_PT
0									
120									

```
type id_no: 0..120;
     dict_tab: array[0..120] of dict_tab_entry;
     dict_tab_entry = record
                        user: array[0..6] of char;
                        context: array[0..9] of id_no;
                        next_seq_no: integer;
                        attribute: 0..4;
                        longevity: 0..5;
                        availability: boolean;
                        copies_no: integer;
                        copyability: boolean;
                        value_ptr: integer
                      end;
```

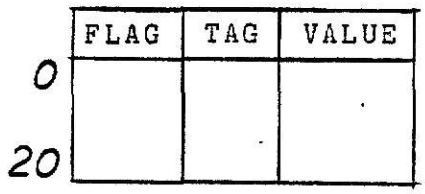
```
attribute:
        1: integer
        2: real
        3: chars
        4: boolean
        0: otherwise

longevity:
        1: fix_tab
        2: static_tab
        3: dynam_tab
        4: fluid_tab
        5: proc_tab
        0: otherwise
```

FIX_TAB:



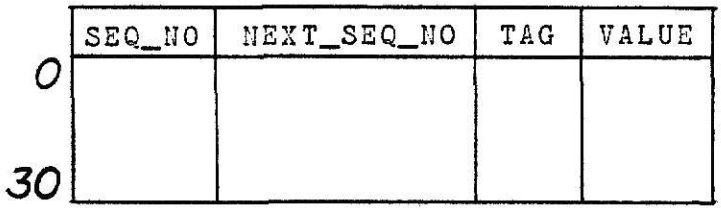

```
type fix_tab : array[0..30] of fix_tab_entry;
     fix_tab_entry : record
                       tag: 0..4;
                       case tag of
                         1: (ivalue: integer);
                         2: (rvalue: real);
                         3: (cvalue: integer);
                            "integer: the index
                             of string_tab"
                         4: (bvalue: boolean)
                     end
```

STATIC_TAB:



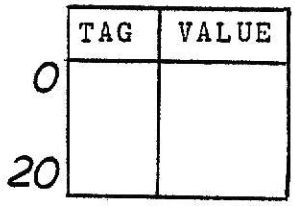

```
type static_tab : array[0..20] of static_tab_entry;
     static_tab_entry = record
                          flag: 0..1; "0: undefined "
                                      "1: defined   "
                          tag: 0..4;
                          case tag of
                            1: (ivalue: integer);
                            2: (rvalue: real);
                            3: (cvalue: integer);
                            4: (bvalue: boolean)
                        end
```

DYNAM_TAB:



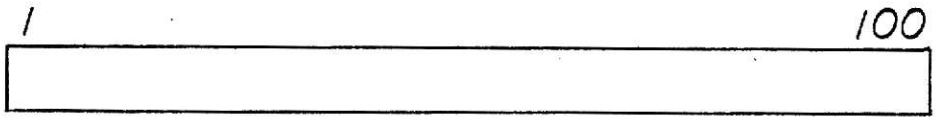

```
type dynam_tab: array[0..30] of dynam_tab_entry;
     dynam_tab_entry = record
                         seq_no: integer;
                         next_seq_no: integer;
                         case tag: 0..4 of
                                 1: (ivalue: integer);
                                 2: (rvalue: real);
                                 3: (cvalue: integer);
                                 4: (bvalue: boolean)
                       end
```

FLUID_TAB:



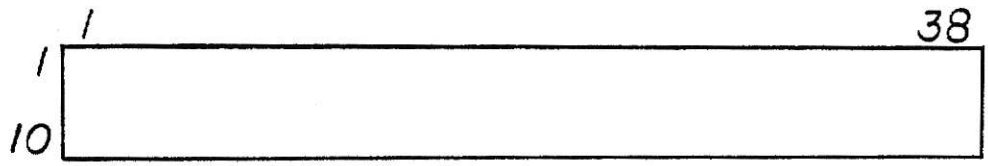

```
type fluid_tab =array[0..20] of fluid_tab_entry;
     fluid_tab_entry = record
                         case tag: 0..4 of
                                 1: (ivalue: integer);
                                 2: (rvalue: real);
                                 3: (cvalue: integer);
                                 4: (bvalue: boolean)
                       end
```

STIMU_TAB:



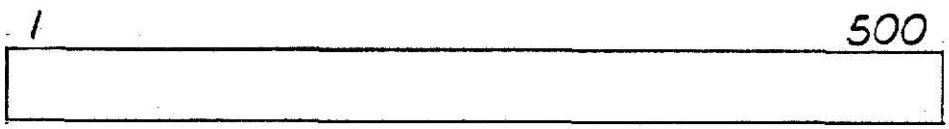

```
type stimu_tab = array[1..100] of char;

"at the end of every stimulation mark ";""
"every object represented  by  dindex(dict_tab's index) "
"or findex (fix_tab's index)  except  abs(    ) and      "
"v_exist(    )                                          "
```

TERMI_TAB:




```
type termi_tab = array[1..10] of termi_tab_entry;
     termi_tab_entry = array[1..38] of char;

initialize termi_tab_entry=
'FOR D(    ):= (    )TO  (    )BY F(    )DO'
```

PROC_TAB:




```
type proc_tab = array[1..500] of char;
```

- proc_tab store target code
- at the end of program mark '.'
- at the end of every procedure mark '!'
- at the end of smart statement mark ';'
- seperate smart statement five component by '$'

for example:

```
$$D11.+0,D12.+0,D13$10$11.+1,12.+1$1;
```

S: stimu_tab's index, if there is no stimulation
S = null

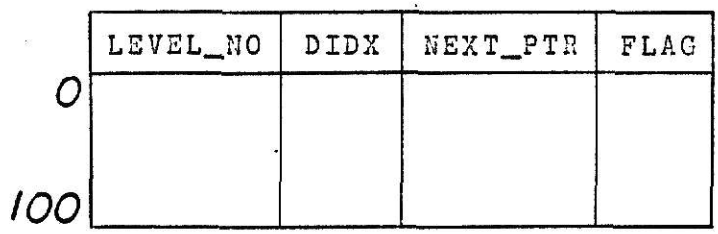
M: input list,D11 means dict_tab's index= 11, F11 means fix_tab's index= 11

A: if it is a procedure name then this part will show the position of it in dict_tab

R: output list, there isn't D or F appear, because this component only cover dict_tab's index

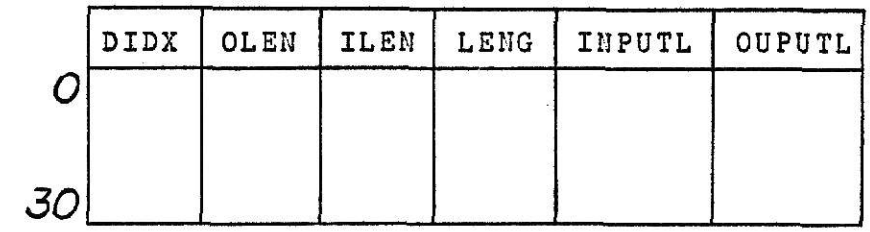
T: the position of termination in termi_tab

ID_TAB:



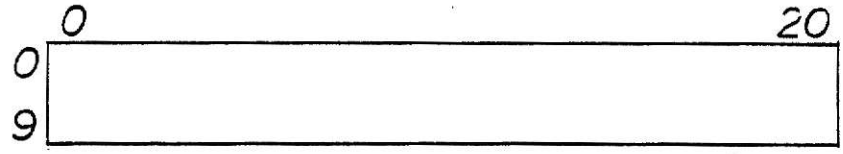

```
type id_tab = array[0..100] of id_tab_entry;
     id_tab_entry = record
                      level_no: integer;
                      didx: integer;
                      next_ptr: integer;
                      flag: 0..1
                    end
```

PID_TAB:




```
type indx_no = 1..120;
     parmlist = array[1..5] of indx_no;
     inparm = record
                flag: char;  "D: dict_tab, F: fix_tab"
                indx: integer
              end;
     AAAAA = array[1..5] of inparm;
     pid_tab = array[0..30] of pid_tab_entry;
     pid_tab_entry = record
                       didx: integer;
                       olen: integer;  "ouputl's no "
                       ilen: integer;  "inputl's no "
                       leng: integer;  "length of proc"
                       inputl: AAAAA;
                       ouputl: parmlist
                     end
```

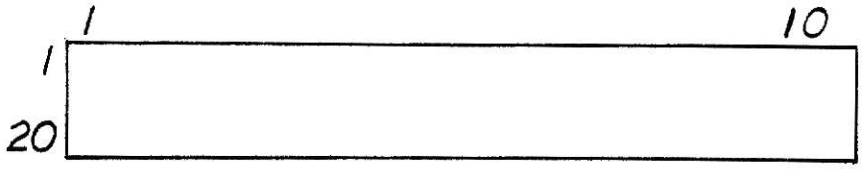
LEVEL_ID_TAB:




```
type level_id_tab = array[0..9] of level_id_entry;
     level_id_entry = array[0..20] of integer;
```

- level_id_tab[I,0] store curnt_coll
- level_id_tab[I,1] store the head line of one procedure
- another parts store hash keys

STRING_TAB:




```
type string_tab = array[1..20] of string_tab_entry
     string_tab_entry = array[1..10] of char;
```

APPENDIX E TRANSLATOR

THIS BOOK CONTAINS NUMEROUS PAGES WITH THE ORIGINAL PRINTING ON THE PAGE BEING CROOKED.

THIS IS THE BEST IMAGE AVAILABLE.

```
1    "ROBERT YOUNG"
2    "KANSAS STATE UNIVERSITY"
3    "DEPARTMENT OF COMPUTER SCIENCE"
4
5    CONST COPYRIGHT = 'COPYRIGHT ROBERT YOUNG 1978';
6
7    "###########
8    #  PREFIX  #
9    ###########"
10
11
12   CONST NL = '(:10:)';   FF = '(:12:)';   CR = '(:13:)';   EM = '(:25:)';
13
14   CONST PAGELENGTH = 512;
15   TYPE PAGE = ARRAY (.1..PAGELENGTH.) OF CHAR;
16
17   CONST LINELENGTH = 132;
18   TYPE LINE = ARRAY (.1..LINELENGTH.) OF CHAR;
19
20   CONST IDLENGTH = 12;
21   TYPE IDENTIFIER = ARRAY (.1..IDLENGTH.) OF CHAR;
22
23   TYPE FILE = 1..2;
24
25   TYPE FILEKIND = (EMPTY, SCRATCH, ASCII, SEQCODE, CONCODE);
26
27   TYPE FILEATTR = RECORD
28                     KIND: FILEKIND;
29                     ADDR: INTEGER;
30                     PROTECTED: BOOLEAN;
31                     NOTUSED: ARRAY (.1..5.) OF INTEGER
32                   END;
33
34   TYPE IODEVICE =
35     (TYPEDEVICE, DISKDEVICE, TAPEDEVICE, PRINTDEVICE, CARDDEVICE);
36
37   TYPE IOOPERATION = (INPUT, OUTPUT, MOVE, CONTROL);
38
39   TYPE IOARG = (WRITEEOF, REWIND, UPSPACE, BACKSPACE);
40
41   TYPE IORESULT =
42     (COMPLETE, INTERVENTION, TRANSMISSION, FAILURE,
43      ENDFILE, ENDMEDIUM, STARTMEDIUM);
44
45   TYPE IOPARAM = RECORD
46                    OPERATION: IOOPERATION;
47                    STATUS: IORESULT;
48                    ARG: IOARG
49                  END;
50
51   TYPE TASKKIND = (INPUTTASK, JOBTASK, OUTPUTTASK);
52
53   TYPE ARGTAG =
54     (NILTYPE, BOOLTYPE, INTTYPE, IDTYPE, PTRTYPE);
55
56   TYPE POINTER = @BOOLEAN;
57
```

```
58    TYPE PASSPTR = @PASSLINK;
59
60    TYPE PASSLINK = RECORD
61      OPTIONS: SET OF CHAR;
62      FILLER1: ARRAY(.1..7.) OF INTEGER;
63      FILLER2: BOOLEAN;
64      RESET_POINT: INTEGER;
65      FILLER3: ARRAY (.1..11.) OF POINTER
66      END;
67
68
69    TYPE ARGTYPE = RECORD
70                     CASE TAG: ARGTAG OF
71                       NILTYPE, BOOLTYPE: (BOOL: BOOLEAN);
72                       INTTYPE: (INT: INTEGER);
73                       IDTYPE: (ID: IDENTIFIER);
74                       PTRTYPE: (PTR: PASSPTR)
75                   END;
76
77    CONST MAXARG = 10;
78    TYPE ARGLIST = ARRAY (.1..MAXARG.) OF ARGTYPE;
79
80    TYPE ARGSEQ = (INP, OUT);
81
82    TYPE PROGRESULT =
83      (TERMINATED, OVERFLOW, POINTERERROR, RANGEERROR, VARIANTERROR,
84       HEAPLIMIT, STACKLIMIT, CODELIMIT, TIMELIMIT, CALLERROR);
85
86    PROCEDURE READ(VAR C: CHAR);
87    PROCEDURE WRITE(C: CHAR);
88
89    PROCEDURE OPEN(F: FILE; ID: IDENTIFIER; VAR FOUND: BOOLEAN);
90    PROCEDURE CLOSE(F: FILE);
91    PROCEDURE GET(F: FILE; P: INTEGER; VAR BLOCK: UNIV PAGE);
92    PROCEDURE PUT(F: FILE; P: INTEGER; VAR BLOCK: UNIV PAGE);
93    FUNCTION LENGTH(F: FILE): INTEGER;
94
95    PROCEDURE MARK(VAR TOP: INTEGER);
96    PROCEDURE RELEASE(TOP: INTEGER);
97
98    PROCEDURE IDENTIFY(HEADER: LINE);
99    PROCEDURE ACCEPT(VAR C: CHAR);
100   PROCEDURE DISPLAY(C: CHAR);
101
102   PROCEDURE NOTUSED;
103   PROCEDURE NOTUSED2;
104   PROCEDURE NOTUSED3;
105   PROCEDURE NOTUSED4;
106   PROCEDURE NOTUSED5;
107   PROCEDURE NOTUSED6;
108
109   PROCEDURE NOTUSED7;
110
111   PROCEDURE NOTUSED8;
112
113   PROCEDURE NOTUUED9;
114
115   PROCEDURE NOTUSED10;
```

```
116
117    PROCEDURE RUN(ID: IDENTIFIER; VAR PARAM: ARGLIST;
118                  VAR LINE: INTEGER; VAR RESULT: PROGRESULT);
119
120
121
122    PROGRAM P(PARAM: LINE);
123    CONST HASH_MAX=50; "500"   ID_HASH_MAX=100; "1000"
124          PINDEX_MAX=30; "300" DINDEX_MAX=120; "800"
125          SPLITLENGTH=2;  "WORDS PER SPLIT REAL"
126          ATTRI_NO=4; LONGE_NO=5;
127          MIN_ORD=0;  MAX_ORD=127;
128          CASECOR=32; SPAN=26;
129          KEY_WORD_NO=14;
130          MAX_INTEGER=2147483647;
131          INTEGER_LIMIT=2147448363;
132          MAX_EXPONENT=38;
133          MAX_STRING_LENGTH=10;
134
135    TYPE V_MARK = 0..1;
136    TYPE ID_TAB_ENTRY = RECORD
137                          LEVEL_NO : INTEGER;
138                          DIDX : INTEGER;
139                          NEXT_PTR : INTEGER;
140                          FLAG : V_MARK
141                        END;
142
143    TYPE ATOK = RECORD
144                  INDEX : INTEGER;
145                  CLAS  : ARRAY[0..6] OF CHAR
146                END;
147
148    TYPE INPARM = RECORD
149                    FLAG : CHAR;
150                    INDX : INTEGER
151                  END;
152
153    TYPE AAAAA = ARRAY [1..5] OF INPARM;
154    TYPE INDX_NO = 1..120;
155         CONTEXT_TYPE = ARRAY[0..9] OF INDX_NO;
156         PARMLIST = ARRAY[1..5] OF INDX_NO;
157         PID_TAB_ENTRY = RECORD
158                           DIDX : INTEGER;
159                           OLEN : INTEGER;
160                           ILEN : INTEGER;
161                           LENG : INTEGER;
162                           INPUTL : AAAAA;
163                           OUPUTL : PARMLIST
164                         END;
165
166    TYPE DICT_TAB_ENTRY = RECORD
167                            USER : ARRAY[0..6] OF CHAR;
168                            CONTEXT : CONTEXT_TYPE;
169                            NEXT_SEQ_NO : INTEGER;
170                            ATTRIBUTE : 0..4;
171                            LONGEVITY : 0..5;
172                            AVAILABILITY : BOOLEAN;
173                            COPIES_NO : INTEGER;
```

```
174                          COPYABILITY : BOOLEAN;
175                          VALUE_PTR : INTEGER
176                        END;
177
178     TYPE VALUETYPE = 0..4;
179          FIX_TAB_ENTRY = RECORD
180                           CASE TAG : VALUETYPE OF
181                                1 : (IVALUE : INTEGER);
182                                2 : (RVALUE : REAL);
183                                3 : (CVALUE : INTEGER);
184                                4 : (BVALUE : BOOLEAN)
185                         END;
186
187     TYPE STATIC_TAB_ENTRY = RECORD
188                              FLAG : V_MARK;
189                              CASE TAG : VALUETYPE OF
190                                   1 : (IVALUE : INTEGER);
191                                   2 : (RVALUE : REAL);
192                                   3 : (CVALUE : INTEGER);
193                                   4 : (BVALUE : BOOLEAN)
194                             END;
195
196     TYPE FLUID_TAB_ENTRY = RECORD
197                             CASE TAG : VALUETYPE OF
198                                  1 : (IVALUE : INTEGER);
199                                  2 : (RVALUE : REAL);
200                                  3 : (CVALUE : INTEGER);
201                                  4 : (BVALUE : BOOLEAN)
202                           END;
203
204     TYPE DYNAMIC_TAB_ENTRY = RECORD
205                               SEQ_NO : INTEGER;
206                               NEXT_SEQ_NO : INTEGER;
207                               CASE TAG : VALUETYPE OF
208                                    1 : (IVALUE : INTEGER);
209                                    2 : (RVALUE : REAL);
210                                    3 : (CVALUE: INTEGER);
211                                    4 : (BVALUE : BOOLEAN)
212                             END;
213
214     TYPE PROC_TABLE = ARRAY[1..500] OF CHAR; "9999"
215          STRING_TAB_ENTRY = ARRAY[1..10] OF CHAR;
216          STRING_TABLE = ARRAY[0..20] OF STRING_TAB_ENTRY; "50"
217          IDHASH_TABLE = ARRAY[0..ID_HASH_MAX] OF ID_TAB_ENTRY;
218          LEVEL_ID_ENTRY = ARRAY[0..20] OF INTEGER;
219          LEVEL_ID_TABLE = ARRAY[0..9] OF LEVEL_ID_ENTRY;
220          PID_TABLE = ARRAY[0..30] OF PID_TAB_ENTRY; "300"
221          DICTIONARY_TABLE = ARRAY[0..120] OF DICT_TAB_ENTRY; "800"
222          FIXED_TABLE = ARRAY[0..30] OF FIX_TAB_ENTRY; "200"
223          STATIC_TABLE = ARRAY[0..20] OF STATIC_TAB_ENTRY; "150"
224          FLUID_TABLE = ARRAY[0..20] OF FLUID_TAB_ENTRY; "100"
225          DYNAMIC_TABLE = ARRAY[0..30] OF DYNAMIC_TAB_ENTRY; "300"
226          CODE2_TYPE = ARRAY[1..38] OF CHAR;
227          TERMI_TABLE = ARRAY[0..10] OF CODE2_TYPE; "30"
228          STIMU_TABLE = ARRAY[1..100] OF CHAR; "999"
229          ID_TYPE = ARRAY[0..6] OF CHAR;
230          KEY_TABLE = ARRAY[1..14] OF ID_TYPE;
231          SYNC_TABLE = ARRAY[1..10] OF ID_TYPE;
```

```
232         ATTRI_TABLE = ARRAY[0..4] OF ID_TYPE;
233         LONGE_TABLE = ARRAY[0..5] OF ID_TYPE;
234         TWOCH = ARRAY[1..2] OF CHAR;
235         COMP_OP_TABLE = ARRAY[1..6] OF TWOCH;
236         OP_TABLE = ARRAY[1..6] OF TWOCH;
237         THREE_ARY = ARRAY[1..3] OF INTEGER;
238
239     TYPE EROR_TYPE = ARRAY[0..19] OF CHAR;
240         CINDEX = ARRAY[1..4] OF CHAR;
241         CLIST = ARRAY[1..25] OF CHAR;
242         WORD = ARRAY[0..6] OF CHAR;
243         ST_TYPE = ARRAY[0..8] OF CHAR;
244         CODE_TYPE = ARRAY[1..70] OF CHAR;
245         STRING_TYPE = ARRAY[1..10] OF CHAR;
246         SPLITREAL = ARRAY[1..SPLITLENGTH] OF INTEGER;
247         SPLIT_INTEGER = ARRAY[1..2] OF SHORTINTEGER;
248
249     "******************** GLOBAL VARIABLES DECLARATION ********************"
250     VAR ID_TAB : IDHASH_TABLE;
251         PID_TAB : PID_TABLE;
252         DICT_TAB : DICTIONARY_TABLE;
253         FIX_TAB : FIXED_TABLE;
254         STATIC_TAB : STATIC_TABLE;
255         FLUID_TAB : FLUID_TABLE;
256         DYNAM_TAB : DYNAMIC_TABLE;
257         PROC_TAB : PROC_TABLE;
258         STRING_TAB : STRING_TABLE;
259         STIMU_TAB : STIMU_TABLE;
260         TERMI_TAB : TERMI_TABLE;
261         COMP_OP_TAB : COMP_OP_TABLE;
262         OP_TAB : OP_TABLE;
263         ATTRI_TAB : ATTRI_TABLE;
264         LONGE_TAB : LONGE_TABLE;
265         TERMI_IDX : INTEGER;
266         STIMU_IDX : INTEGER;
267         CHAR_INDEX : INTEGER;
268         STRING_TEXT : STRING_TYPE;
269         STRING_LENGTH : INTEGER;
270         STRING_COUNTER : INTEGER;
271         CURRENT_CONTEXT : CONTEXT_TYPE;
272         FIX_COUNTER,DYN_COUNTER : INTEGER;
273         STA_COUNTER,FLD_COUNTER : INTEGER;
274         CH:CHAR; AUNIT:ATOK; KK:INTEGER;
275         CURRENT_PIDX : INTEGER;
276         CURRENT_DIDX : INDX_NO;
277         CURRENT_LEVEL : INTEGER;
278         CSNO,CANO,CTNO : CINDEX;
279         CRNO,CMNO : CLIST;
280         CTMP : CODE_TYPE;
281         CURNT_ST_NO : INTEGER;
282         CURNT_COLL : INTEGER;
283         ERRCODE : BOOLEAN;
284         TEMP_CODE : CODE_TYPE;
285         CODEIDX : INTEGER;
286         EROR_TEXT : EROR_TYPE;
287         KEY_TAB : KEY_TABLE;
288         MK0,MK1:V_MARK; P_TOP:0..9;
289         P_STACK : ARRAY[0..9] OF INTEGER;
```

```
290       DY_STACK : ARRAY[0..9] OF INTEGER;
291       SYNC_TAB : SYNC_TABLE;
292       ID_TEXT : ID_TYPE;
293       BLANK : ID_TYPE;
294       BLANK1 : ARRAY[1..10] OF CHAR;
295       BLANK2 : ARRAY[1..25] OF CHAR;
296       REAL0,REAL1,REAL10,REAL_LIMIT,MAX_REAL : REAL;
297       NON_ALFAS,DIGITS,LETTERS,SETS,ALFAMERIC:SET OF CHAR;
298       PROGRAM_HEAD : BOOLEAN;
299       EMPTY0 : THREE_ARY;
300       AHKY : INTEGER;
301       LEVEL_ID_TAB : LEVEL_ID_TABLE;
302
303
304   "*************************************************************************
305   *  SPLIT_ASIGN:                                                         *
306   *    CREATE INTEGER CONSTANTS BIGGER THAN 32K FROM TWO SHORTINTEGER     *
307   *    CONSTANTS, THIS ALLOWS USE OF LARGE CONSTANTS WHILE COMPILING      *
308   *    ON A 16_BIT SOURCE MACHINE                                         *
309   *************************************************************************"
310
311   PROCEDURE SPLIT_ASSIGN(VAR TARGET:UNIV SPLIT_INTEGER;WD1,WD2:INTEGER);
312     BEGIN
313       TARGET[1]:=WD1;
314       TARGET[2]:=WD2
315     END;
316
317
318   "***************** LARGEST_REAL:TO GET THE MAX_REAL ******************"
319   PROCEDURE LARGEST_REAL(VAR MAX:UNIV SPLITREAL);
320     BEGIN
321       SPLIT_ASSIGN(MAX[1],32767,-1);  "7FFFFFFF"
322       MAX[1]:=-1
323     END;
324
325
326   "***************** WRITESTMT: WRITE 'STATEMENT' ***********************"
327   PROCEDURE WRITESTMT(ST:ST_TYPE);
328     VAR I : INTEGER;
329     BEGIN
330       WRITE(NL);
331       FOR I := 0 TO 8 DO WRITE(ST[I]);
332       WRITE(' ')
333     END;
334
335
336   "***************** ERROR:WRITE ERROR MESSAGE **************************"
337   PROCEDURE ERROR(ST_NO:INTEGER;TEXT:EROR_TYPE);
338     VAR T : ARRAY[1..4] OF CHAR;
339         REM,DIGIT,I : INTEGER;
340     BEGIN
341       REM := ST_NO;
342       DIGIT := 0;
343       REPEAT
344         DIGIT := DIGIT + 1;
345         T[DIGIT] := CHR((REM MOD 10) + ORD('0'));
346         REM := REM DIV 10
347       UNTIL (REM=0);
```

```
348       WRITESTMT('STATEMENT');
349       FOR I:= DIGIT DOWNTO 1 DO WRITE(T[I]);
350       FOR I:= DIGIT+1 TO 5 DO WRITE(' ');
351       FOR I:= 0 TO 19 DO WRITE(TEXT[I]);
352       WRITE(NL)
353     END;
354
355
356   "********** CONV_NUM:CONVERT INTEGER TO ARRAY[1..4] OF CHAR **********"
357   PROCEDURE CONV_NUM(VAR CHDX:CINDEX;ANUM:INTEGER);
358     VAR ATEMP:CINDEX; I,L,ANOM:INTEGER;
359     BEGIN
360       L:=0; CHDX:='    ';
361       ANOM:=ANUM;
362       REPEAT
363         L:=L + 1;
364         ATEMP[L]:=CHR((ANOM MOD 10) + ORD('0'));
365         ANOM:=(ANOM DIV 10)
366       UNTIL (ANOM = 0) OR (L=4);
367       I:=0;
368       REPEAT
369         I:=I + 1;
370         CHDX[I]:=ATEMP[L];
371         L:=L - 1
372       UNTIL (L=0)
373     END;
374
375
376   "********************* CHECK_NUM ***************************************"
377   PROCEDURE CHECK_NUM(NUM:INTEGER);
378     VAR CHIX:CINDEX; I:INTEGER;
379     BEGIN
380       CONV_NUM(CHIX,NUM);
381       FOR I:=1 TO 5 DO WRITE('*'); WRITE(' ');
382       FOR I:=1 TO 4 DO WRITE(CHIX[I]); WRITE(NL)
383     END;
384
385
386   "************************ CHECK_NUM1 *********************************"
387   PROCEDURE CHECK_NUM1(NUM:INTEGER);
388     VAR CHIX:CINDEX; I:INTEGER;
389     BEGIN
390       CONV_NUM(CHIX,NUM);
391       FOR I:=1 TO 4 DO WRITE(CHIX[I])
392     END;
393
394
395   "************************ INITIALIZE ***********************************"
396   PROCEDURE INITIALIZE;
397     VAR C:MIN_ORD..MAX_ORD;
398         I : INTEGER;
399         BKINPARM : AAAAA;
400         BKCONTEXT: CONTEXT_TYPE;
401         BKPARM: PARMLIST;
402         BLANK3: CODE2_TYPE;
403         ALID_ENTRY : LEVEL_ID_ENTRY;
404     BEGIN
405       REAL0:=CONV(0);
```

```
406         REAL1:=CONV(1);
407         REAL10:=CONV(10);
408         BLANK:='       ';
409         BLANK1:='          ';
410         BLANK2:='                          ';
411         BLANK3:='                                      ';
412         FOR I:=1 TO 3 DO EMPTY0[I]:=0;
413         CURRENT_LEVEL := 0;
414         CURRENT_DIDX := 1;
415         CODEIDX:=0; P_TOP:=0;
416         PROGRAM_HEAD:=TRUE;
417         FIX_COUNTER := 1;
418         DYN_COUNTER := 1;
419         STA_COUNTER := 1;
420         FLD_COUNTER := 1;
421         STRING_COUNTER := 1;
422         CURNT_ST_NO := 1;
423         STIMU_IDX := 1;
424         CURNT_COLL := 51; "501"
425         DIGITS := ['0','1','2','3','4','5','6','7','8','9'];
426         LETTERS := ['A','B','C','D','E','F','G','H','I','J','K','L','M',
427                     'N','O','P','Q','S','T','U','V','W','X','Y','Z','_',
428                     'a','b','c','d','e','f','g','i','j','k','l','m','n',
429                     'o','p','q','r','S','t','u','v','w','x','y','z','R'];
430         ALFAMERIC := LETTERS OR DIGITS;
431         NON_ALFAS := [];
432         FOR C:=MIN_ORD TO MAX_ORD DO NON_ALFAS:=NON_ALFAS OR [CHR(C)];
433         NON_ALFAS := NON_ALFAS - ALFAMERIC;
434
435         "***************** INITIALIZE TABLES ******************************"
436         FOR I:=0 TO 20 DO ALID_ENTRY[I]:=0;
437         FOR I:=1 TO 5 DO BKPARM[I]:=0;
438         FOR I:=0 TO 9 DO BKCONTEXT[I]:=0;
439         FOR I:=1 TO 5 DO BEGIN
440             BKINPARM[I].FLAG:=' ';
441             BKINPARM[I].INDX:=0
442             END;
443
444         "*LEVEL_ID_TAB*"
445         FOR I:=0 TO 9 DO LEVEL_ID_TAB[I]:=ALID_ENTRY;
446
447         "*STRING_TAB*"
448         FOR I:=0 TO 20 DO STRING_TAB[I]:=BLANK1; "50"
449
450         "*STIMU_TAB*"
451         FOR I:=1 TO 100 DO STIMU_TAB[I]:=' '; "999"
452
453         "*TERMI_TAB*"
454         FOR I:=0 TO 10 DO TERMI_TAB[I]:=BLANK3; "30"
455
456         "*PROC_TAB*"
457         FOR I:=1 TO 500 DO PROC_TAB[I]:=' '; "9999"
458
459         "*DICT_TAB*"
460         FOR I:=0 TO 120 DO BEGIN "800"
461             DICT_TAB[I].USER:=BLANK;
462             DICT_TAB[I].CONTEXT:=BKCONTEXT;
463             DICT_TAB[I].NEXT_SEQ_NO:=0;
```

```
464             DICT_TAB[I].ATTRIBUTE:=0;
465             DICT_TAB[I].LONGEVITY:=0;
466             DICT_TAB[I].AVAILABILITY:=TRUE;
467             DICT_TAB[I].COPIES_NO:=0;
468             DICT_TAB[I].COPYABILITY:=TRUE;
469             DICT_TAB[I].VALUE_PTR:=0
470             END;
471
472         "*PID_TAB*"
473         FOR I:=0 TO 30 DO BEGIN "300"
474             PID_TAB[I].DIDX:=0; PID_TAB[I].OLEN:=0;
475             PID_TAB[I].ILEN:=0; PID_TAB[I].LENG:=0;
476             PID_TAB[I].INPUTL:=BKINPARM;
477             PID_TAB[I].OUPUTL:=BKPARM
478             END;
479
480         "*ID_TAB*"
481         FOR I:=0 TO 100 DO BEGIN "1000"
482             ID_TAB[I].LEVEL_NO:=0; ID_TAB[I].DIDX:=0;
483             ID_TAB[I].NEXT_PTR:=0; ID_TAB[I].FLAG:=0
484             END;
485
486         "*FIX_TAB*"
487         FOR I:=0 TO 30 DO BEGIN "200"
488             FIX_TAB[I].TAG:=1; FIX_TAB[I].IVALUE:=0
489             END;
490
491         "*STATIC_TAB*"
492         FOR I:=0 TO 20 DO BEGIN "150"
493             STATIC_TAB[I].FLAG:=0; STATIC_TAB[I].TAG:=1;
494             STATIC_TAB[I].IVALUE:=0
495             END;
496
497         "*DYNAM_TAB*"
498         FOR I:=0 TO 30 DO BEGIN "300"
499             DYNAM_TAB[I].SEQ_NO:=0; DYNAM_TAB[I].NEXT_SEQ_NO:=0;
500             DYNAM_TAB[I].TAG:=1; DYNAM_TAB[I].IVALUE:=0
501             END;
502
503         "*FLUID_TAB*"
504         FOR I:=0 TO 20 DO BEGIN "100"
505             FLUID_TAB[I].TAG:=1; FLUID_TAB[I].IVALUE:=0
506             END;
507
508         "*KEY_TAB*"
509         KEY_TAB[1]:='PROGRAM'; KEY_TAB[2]:='VAR    ';
510         KEY_TAB[3]:='CONT   '; KEY_TAB[4]:='END    ';
511         KEY_TAB[5]:='AND    '; KEY_TAB[6]:='OR     ';
512         KEY_TAB[7]:='NOT    '; KEY_TAB[8]:='ABS    ';
513         KEY_TAB[9]:='V_EXIST'; KEY_TAB[10]:='TRUE   ';
514         KEY_TAB[11]:='FALSE  '; KEY_TAB[12]:='BEGIN  ';
515         KEY_TAB[13]:='INC    '; KEY_TAB[14]:='       ';
516
517         "*OP_TAB*"
518         OP_TAB[1]:='+ '; OP_TAB[2]:='- ';
519         OP_TAB[3]:='/ '; OP_TAB[4]:='* ';
520         OP_TAB[5]:='**'; OP_TAB[6]:=':=';
521
```

```
522         "*COMP_OP_TAB*"
523         COMP_OP_TAB[1]:='= '; COMP_OP_TAB[2]:='> ';
524         COMP_OP_TAB[3]:='>='; COMP_OP_TAB[4]:='< ';
525         COMP_OP_TAB[5]:='<='; COMP_OP_TAB[6]:='<>';
526
527         "*LONGE_TAB*"
528         LONGE_TAB[1]:='FIX    ';  LONGE_TAB[2]:='STATIC ';
529         LONGE_TAB[3]:='DYNAMIC';  LONGE_TAB[4]:='FLUID  ';
530         LONGE_TAB[5]:='PROC   ';  LONGE_TAB[0]:='       ';
531
532         "*ATTRI_TAB*"
533         ATTRI_TAB[1]:='INTEGER';  ATTRI_TAB[2]:='REAL   ';
534         ATTRI_TAB[3]:='CHAR   ';  ATTRI_TAB[4]:='BOOLEAN';
535         ATTRI_TAB[0]:='       ';
536         LARGEST_REAL(MAX_REAL);
537         REAL_LIMIT := MAX_REAL/REAL10
538       END;
539
540
541     "******** INSERT_ID:DICT_TAB(USER,CONTEXT),ID_TAB(LEVEL_NO,DIDX) ******"
542     PROCEDURE INSERT_ID(AID_TEXT:ID_TYPE;II:INTEGER);
543       BEGIN
544         ID_TAB[II].LEVEL_NO := CURRENT_LEVEL;
545         ID_TAB[II].DIDX := CURRENT_DIDX;
546         DICT_TAB[ID_TAB[II].DIDX].USER := AID_TEXT;
547         DICT_TAB[ID_TAB[II].DIDX].CONTEXT := CURRENT_CONTEXT;
548         CURRENT_DIDX := SUCC(CURRENT_DIDX)
549       END;
550
551
552     "*************************************************************************
553     *  SEARCH_ID(AID_TEXT:ID_TYPE;MMKK:V_MARK;VAR I:INTEGER);                *
554     *    1. (MMKK=1):DECLARE ID, (MMKK=0):REFERENCE ID.                      *
555     *    2. SEARCH ID_TAB AND CALL INSERT_ID                                 *
556     *    3. CATCH OUT UNDEFINED ID AND REDEFINED ID                          *
557     *************************************************************************"
558
559     PROCEDURE SEARCH_ID(AID_TEXT:ID_TYPE;MMKK:V_MARK;VAR I:INTEGER);
560       VAR FINISHED:BOOLEAN; IL,IK:INTEGER;
561           TEMP:ID_TAB_ENTRY; MIL:INTEGER;
562       BEGIN
563         FINISHED:=FALSE; IL:=I; TEMP.FLAG:=0;
564         TEMP.LEVEL_NO:=0; TEMP.DIDX:=0; TEMP.NEXT_PTR:=0;
565         CASE MMKK OF
566           1 : REPEAT
567                 IF (ID_TAB[IL].DIDX=0) THEN BEGIN
568                    INSERT_ID(AID_TEXT,IL);
569                    FINISHED:=TRUE; I:=IL
570                    END ELSE
571                    IF (ID_TAB[IL].LEVEL_NO=CURRENT_LEVEL) &
572                       (DICT_TAB[ID_TAB[IL].DIDX].USER=AID_TEXT)
573                       THEN BEGIN
574                       IF (ID_TAB[IL].FLAG=1)
575                          THEN ERROR(CURNT_ST_NO,'REDEFINED IDENTIFIER');
576                       FINISHED:=TRUE; I:=IL
577                       END ELSE BEGIN
578                       IK:=ID_TAB[IL].NEXT_PTR;
579                       IF (IK=0) THEN BEGIN
```

```
580                         ID_TAB[IL].NEXT_PTR:=CURNT_COLL;
581                         IL:=CURNT_COLL;
582                         CURNT_COLL:=SUCC(CURNT_COLL)
583                         END ELSE IL:=IK
584                       END
585               UNTIL(FINISHED);
586           0 : IF (ID_TAB[IL].DIDX=0) THEN BEGIN
587                 ERROR(CURNT_ST_NO,'UNDEFINED IDENTIFIER');
588                 FINISHED:=TRUE; I:=IL
589                 END ELSE BEGIN
590                 REPEAT
591                   IF (DICT_TAB[ID_TAB[IL].DIDX].USER=AID_TEXT)
592                      THEN BEGIN
593                      MIL:=IL; TEMP:=ID_TAB[IL] END;
594                   IL:=ID_TAB[IL].NEXT_PTR
595                 UNTIL(IL=0);
596                 IF (TEMP.FLAG=0) THEN
597                    ERROR(CURNT_ST_NO,'UNDEFINED IDENTIFIER');
598                 I:=MIL
599                 END;
600         END"CASE"
601     END;
602
603
604   "*************************************************************************
605   *  INSERT_PID:                                                          *
606   *    1.INSERT PID_TAB(DIDX),DICT_TAB(USER,CONTEXT,LONGEVITY)            *
607   *    2.FILL UP CURRENT_CONTEXT                                          *
608   *************************************************************************"
609
610   PROCEDURE INSERT_PID(AID_TEXT:ID_TYPE;L:INTEGER);
611     BEGIN
612       PID_TAB[L].DIDX := CURRENT_DIDX;
613       DICT_TAB[CURRENT_DIDX].USER := AID_TEXT;
614       DICT_TAB[CURRENT_DIDX].CONTEXT := CURRENT_CONTEXT;
615       DICT_TAB[CURRENT_DIDX].LONGEVITY := 5;
616       CURRENT_CONTEXT[CURRENT_LEVEL] := PID_TAB[L].DIDX;
617       CURRENT_DIDX := SUCC(CURRENT_DIDX)
618     END;
619
620
621   "*************************************************************************
622   *  SEARCH_PID(IDTEXT:ID_TYPE;BBMARK:V_MARK;VAR I:INTEGER);              *
623   *    1. (BBMARK=1):DEFINE PROCEDURE, (BBMARK=0):REFERENCE PROCEDURE     *
624   *    2. SEARCH PID_TAB AND CALL INSERT_PID                              *
625   *    3. FIND OUT REDEFINED PROCEDURE & UNDEFINED PROCEDURE              *
626   *************************************************************************"
627
628   PROCEDURE SEARCH_PID(IDTEXT:ID_TYPE;BBMARK:V_MARK;VAR I:INTEGER);
629     VAR FINISHED:BOOLEAN;
630     BEGIN
631       I:=1; FINISHED:=FALSE;
632       REPEAT
633         CASE BBMARK OF
634          1 : IF (PID_TAB[I].DIDX=0) THEN BEGIN
635                INSERT_PID(IDTEXT,I);
636                FINISHED:=TRUE;
637                CURRENT_PIDX:=I
```

```
638                 END ELSE
639                 IF (DICT_TAB[PID_TAB[I].DIDX].USER=IDTEXT) THEN BEGIN
640                    ERROR(CURNT_ST_NO,'REDEFINED PROC_NAME ');
641                    FINISHED:=TRUE
642                    END ELSE I:=I+1;
643           0 : IF (PID_TAB[I].DIDX=0) THEN BEGIN
644                 ERROR(CURNT_ST_NO,'UNDEFINED PROC_NAME ');
645                 FINISHED:=TRUE; I:=0
646                 END ELSE
647                 IF (DICT_TAB[PID_TAB[I].DIDX].USER=IDTEXT)
648                    THEN FINISHED:=TRUE
649                    ELSE I:=I+1
650         END"CASE"
651       UNTIL (FINISHED)
652     END;
653
654
655   "********** SEARCH_TABS:SEARCH KEY_TAB,ATTRI_TAB,LONGE_TAB **********"
656   PROCEDURE SEARCH_TABS(AID_TEXT:ID_TYPE;VAR L:INTEGER;
657                         VAR SVALUE:BOOLEAN);
658     VAR I : INTEGER;
659     BEGIN
660       SVALUE:=FALSE; I:=1;
661       REPEAT
662         IF (KEY_TAB[I]=AID_TEXT) THEN BEGIN
663            L:=I; SVALUE:=TRUE
664            END ELSE I:=I+1
665       UNTIL (SVALUE=TRUE) OR (I>KEY_WORD_NO);
666       IF (SVALUE=FALSE) THEN BEGIN
667          I:=1;
668          REPEAT
669            IF (ATTRI_TAB[I]=AID_TEXT) THEN BEGIN
670               L:=I; SVALUE:=TRUE
671               END ELSE I:=I+1
672          UNTIL (SVALUE=TRUE) OR (I>ATTRI_NO)
673          END;
674       IF (SVALUE=FALSE) THEN BEGIN
675          I:=1;
676          REPEAT
677            IF (LONGE_TAB[I]=AID_TEXT) THEN BEGIN
678               L:=I; SVALUE:=TRUE
679               END ELSE I:=I+1
680          UNTIL (SVALUE=TRUE) OR (I>LONGE_NO)
681          END
682     END;
683
684
685   "*****************************************************************
686   *  IDENTIFIER(AMARK,BMARK);                                      *
687   *    1. (AMARK=1):PROCEDURE IDENTIFIER & CALL SEARCH_PID         *
688   *       (AMARK=0):IDENTIFIER & CALL SEARCH_ID.                   *
689   *    2. (BMARK=1):DEFINE IDENTIFIER,(BMARK=0):REFERENCE IDENTIFIER *
690   *****************************************************************"
691
692   PROCEDURE GET_TOK(VAR UNIT:ATOK;MARK0,MARK1:V_MARK;VAR HHKY:INTEGER);
693             FORWARD;
694   PROCEDURE IDENTIFIER(AMARK,BMARK:V_MARK;VAR HKY:INTEGER);
695     VAR UCCH : CHAR;
```

```
696         SEARCHIT : BOOLEAN;
697         ERR_MARK : BOOLEAN;
698         ID_TEXT : ID_TYPE;
699         HASH_KEY : INTEGER;
700         PIDX,IDX,LL,I : INTEGER;
701       BEGIN
702         ID_TEXT := '       '; ERR_MARK:=FALSE;
703         CHAR_INDEX := 0;
704         REPEAT
705           IF (CHAR_INDEX>6) THEN
706              ERR_MARK:=TRUE ELSE BEGIN
707              ID_TEXT[CHAR_INDEX]:=CH;
708              CHAR_INDEX:=SUCC(CHAR_INDEX)
709              END;
710           WRITE(CH); READ(CH)
711         UNTIL (CH IN NON_ALFAS);
712         IF (ERR_MARK=TRUE) THEN BEGIN
713            ERROR(CURNT_ST_NO,'NAME TOO LONG       ');
714            AUNIT.CLAS := '********';
715            AUNIT.INDEX := 0; GET_TOK(AUNIT,MK0,MK1,AHKY)
716            END ELSE BEGIN
717            SEARCH_TABS(ID_TEXT,LL,SEARCHIT);
718            IF (SEARCHIT=TRUE) THEN BEGIN
719               AUNIT.CLAS:=ID_TEXT;
720               AUNIT.INDEX:=LL
721               END ELSE
722               IF (AMARK=1) THEN BEGIN
723                  SEARCH_PID(ID_TEXT,BMARK,PIDX);
724                  AUNIT.CLAS:='PID    ';
725                  AUNIT.INDEX:=PIDX
726                  END ELSE BEGIN
727                  HASH_KEY:=1;
728                  FOR I:=0 TO CHAR_INDEX-1 DO BEGIN
729                      UCCH:=ID_TEXT[I];
730                      HASH_KEY:=HASH_KEY*(ORD(UCCH) MOD SPAN + 1)
731                                MOD HASH_MAX + 1
732                      END;
733                  HKY:=HASH_KEY;
734                  IDX:=HASH_KEY;
735                  "CHECK_NUM(CURRENT_LEVEL);
736                  CHECK_NUM(IDX);"
737                  SEARCH_ID(ID_TEXT,BMARK,IDX);
738                  "CHECK_NUM(IDX); CHECK_NUM(ID_TAB[IDX].NEXT_PTR);"
739                  AUNIT.CLAS:='ID     ';
740                  AUNIT.INDEX:=IDX
741                  END
742            END
743       END;
744
745
746     "********************** SEARCH_FIX_TAB ********************************"
747     PROCEDURE SEARCH_FIX_TAB(VAR I:INTEGER;FIXREC:FIX_TAB_ENTRY);
748       VAR FINISHED : BOOLEAN;
749       BEGIN
750         I:=1; FINISHED:=FALSE;
751         IF (FIX_COUNTER>1) THEN
752            REPEAT
753              IF (FIXREC.TAG = FIX_TAB[I].TAG) THEN
```

```
754                CASE FIXREC.TAG OF
755                  1 : IF (FIXREC.IVALUE = FIX_TAB[I].IVALUE)
756                        THEN FINISHED := TRUE ELSE I:=I+1;
757                  2 : IF (FIXREC.RVALUE = FIX_TAB[I].RVALUE)
758                        THEN FINISHED := TRUE ELSE I:=I+1;
759                  3 : IF (FIXREC.CVALUE = FIX_TAB[I].CVALUE)
760                        THEN FINISHED := TRUE ELSE I:=I+1;
761                  4 : IF (FIXREC.BVALUE = FIX_TAB[I].BVALUE)
762                        THEN FINISHED := TRUE ELSE I:=I+1
763                END"CASE" ELSE I:=I+1
764            UNTIL (FINISHED) OR (I = FIX_COUNTER)
765      END;
766
767
768    "******** NUMBER:AFTER CONSTRUCTION NUMBER CALL SEARCH_FIX_TAB ********"
769    PROCEDURE NUMBER;
770      VAR MANTISSA:REAL; RESULT:REAL;
771          POWER_OF_TEN : REAL;
772          ERROR_SW : BOOLEAN;
773          EXPONENT_SIGN : BOOLEAN;
774          EXPONENT,I,IL : INTEGER;
775          EXPONENT_PART : INTEGER;
776          V_TYPE : V_MARK;
777          ARECORD : FIX_TAB_ENTRY;
778          RESULT1 : INTEGER;
779      BEGIN
780        V_TYPE := 0;
781        MANTISSA := REAL0;
782        ERROR_SW := FALSE;
783        EXPONENT := 0;
784
785        "*COLLECT INTEGER PART*"
786        REPEAT
787          IF (MANTISSA <= "REAL_LIMIT" 32767.0)
788             THEN MANTISSA := MANTISSA*REAL10 + CONV(ORD(CH)-ORD('0'))
789             ELSE ERROR_SW := TRUE;
790          WRITE(CH); READ(CH);
791        UNTIL NOT(CH IN DIGITS);
792
793        "*COLLECT FRACTION PART*"
794        IF (CH = '.') THEN BEGIN
795           WRITE(CH); READ(CH);
796           V_TYPE := 1;
797           IF NOT(CH IN DIGITS)
798              THEN ERROR_SW := TRUE"(CURNT_ST_NO,'NUMBER_ERROR        ')"
799              ELSE REPEAT
800                     IF (MANTISSA <= REAL_LIMIT) THEN BEGIN
801                        MANTISSA:=MANTISSA*REAL10+CONV(ORD(CH)-ORD('0'));
802                        EXPONENT := EXPONENT - 1
803                        END ELSE ERROR_SW:=TRUE;
804                     WRITE(CH); READ(CH)
805                   UNTIL NOT(CH IN DIGITS)
806           END;
807
808        "*COLLECT EXPONENT PART*"
809        IF (CH = 'E') THEN BEGIN
810           V_TYPE := 1;
811           WRITE(CH); READ(CH);
```

```
812          EXPONENT_PART := 0;
813          EXPONENT_SIGN := FALSE;
814          IF (CH = '+') THEN BEGIN
815             WRITE(CH); READ(CH)
816             END ELSE
817             IF (CH = '-') THEN BEGIN
818                EXPONENT_SIGN := TRUE;
819                WRITE(CH); READ(CH)
820                END;
821          IF NOT(CH IN DIGITS)
822             THEN ERROR_SW := TRUE"(CURNT_ST_NO,'NUMBER_ERROR        ')"
823             ELSE REPEAT
824                    IF (EXPONENT_PART <= INTEGER_LIMIT)
825                       THEN EXPONENT_PART:=EXPONENT_PART*10-
826                                           ORD('0') + ORD(CH)
827                       ELSE ERROR_SW := TRUE;
828                  WRITE(CH); READ(CH)
829                UNTIL NOT(CH IN DIGITS);
830          IF (EXPONENT_SIGN)
831             THEN IF (MAX_EXPONENT + EXPONENT >= EXPONENT_PART)
832                     THEN EXPONENT := EXPONENT-EXPONENT_PART
833                     ELSE ERROR_SW := TRUE
834             ELSE EXPONENT := EXPONENT + EXPONENT_PART
835          END;
836
837       "*CONSTRUCT THE NUMBER*"
838       IF (V_TYPE = 0) THEN BEGIN
839          IF (MANTISSA > CONV(MAX_INTEGER)) THEN BEGIN
840             ERROR_SW := TRUE;"(CURNT_ST_NO,'NUMBER_ERROR        ')"
841             MANTISSA := REAL0
842             END;
843          RESULT1 := TRUNC(MANTISSA)
844          END ELSE BEGIN
845          IF (ERROR_SW=TRUE) THEN BEGIN
846             ERROR(CURNT_ST_NO,'NUMBER_ERROR        ');
847             RESULT := REAL0
848             END ELSE BEGIN
849             POWER_OF_TEN := REAL1;
850             IF (EXPONENT < 0) THEN BEGIN
851                EXPONENT_SIGN := TRUE;
852                EXPONENT := ABS(EXPONENT)
853                END ELSE EXPONENT_SIGN := FALSE;
854             IF (EXPONENT > MAX_EXPONENT) THEN BEGIN
855                ERROR(CURNT_ST_NO,'NUMBER_ERROR        ');
856                EXPONENT := 0
857                END
858           END;
859         FOR I:=1 TO EXPONENT DO POWER_OF_TEN:=POWER_OF_TEN*REAL10;
860         "*EITHER MANTISSA=0.0 OR MANTISSA>=1.0*"
861         IF (MANTISSA = REAL0)
862            THEN RESULT:=REAL0
863            "*IF MANTISSA>=1.0 THEN WE MUST HAVE:MANTISSA*POWER_OF_TEN*"
864            "*<=MAX_REAL I.E.POWER_OF_TEN<=MAX_REAL/MANTISSA<=MAX_REAL*"
865            ELSE IF (EXPONENT_SIGN)
866                    THEN RESULT := MANTISSA/POWER_OF_TEN
867                    ELSE IF (POWER_OF_TEN <= MAX_REAL/MANTISSA)
868                            THEN RESULT := MANTISSA*POWER_OF_TEN
869                            ELSE BEGIN
```

```
870                          ERROR(CURNT_ST_NO, 'NUMBER_ERROR        ');
871                          RESULT := REAL0
872                          END
873        END;
874
875      "** CREATE A RECORD : FIX_TAB'S ENTRY **"
876      CASE V_TYPE OF
877           0 : BEGIN
878                 ARECORD.TAG := 1;
879                 ARECORD.IVALUE := RESULT1;
880               END;
881           1 : BEGIN
882                 ARECORD.TAG := 2;
883                 ARECORD.RVALUE := RESULT
884               END
885      END;"CASE"
886
887      "*SEARCH AND INSERT THIS RECORD TO FIX_TAB*"
888      SEARCH_FIX_TAB(IL,ARECORD);
889      IF (IL = FIX_COUNTER) THEN BEGIN
890         FIX_TAB[IL].TAG := ARECORD.TAG;
891         CASE V_TYPE OF
892              0 : FIX_TAB[IL].IVALUE := ARECORD.IVALUE;
893              1 : FIX_TAB[IL].RVALUE := ARECORD.RVALUE
894         END;"CASE"
895         FIX_COUNTER := SUCC(FIX_COUNTER)
896         END;"IF"
897      AUNIT.CLAS := 'CONST  ';
898      AUNIT.INDEX := IL
899    END;
900
901
902    "********* STRING_CHAR:TEST ITS LENGTH & BUILT UP STRING_TEXT ********"
903    PROCEDURE STRING_CHAR(VAR ERR_MARK:BOOLEAN);
904      BEGIN
905        ERR_MARK := FALSE;
906        IF (STRING_LENGTH >= MAX_STRING_LENGTH)
907           THEN BEGIN
908           ERR_MARK:=TRUE;
909           STRING_LENGTH:=SUCC(STRING_LENGTH)
910           END ELSE BEGIN
911           STRING_LENGTH:=SUCC(STRING_LENGTH);
912           STRING_TEXT[STRING_LENGTH]:=CH
913           END;
914        WRITE(CH); READ(CH)
915      END;
916
917
918    "********* SEARCH_STRING_TAB:SEARCH AND INSERT STRING_TEXT ***********"
919    PROCEDURE SEARCH_STRING_TAB(VAR I:INTEGER);
920      VAR FINISHED : BOOLEAN;
921      BEGIN
922        I := 1;
923        REPEAT
924          IF (STRING_TEXT = STRING_TAB[I])
925             THEN FINISHED := TRUE
926             ELSE IF (STRING_TAB[I] <> BLANK1)
927                     THEN I := I + 1
```

```
928       UNTIL (FINISHED) OR (I = STRING_COUNTER);
929       IF (I = STRING_COUNTER)
930          THEN BEGIN
931                 STRING_TAB[I] := STRING_TEXT;
932                 STRING_COUNTER := SUCC(STRING_COUNTER)
933               END;
934     END;
935
936
937   "******* STRING:CALL STRING_CHAR,SEARCH_STRING_TAB,SEACH_FIX_TAB *****"
938   PROCEDURE STRING;
939     VAR  II,IL : INTEGER;
940          ERROR_MARK : BOOLEAN;
941          ASTRING : FIX_TAB_ENTRY;
942     BEGIN
943       STRING_LENGTH := 0;
944       WRITE(CH); READ(CH);
945       WHILE (NOT (CH IN ['''',NL,EM])) DO STRING_CHAR(ERROR_MARK);
946       CASE CH OF
947            '''' : BEGIN
948                      WRITE(CH); READ(CH)
949                   END;
950           NL,EM : ERROR_MARK:=TRUE;
951       END "CASE";
952       IF (ERROR_MARK=FALSE) THEN BEGIN
953          SEARCH_STRING_TAB(II);
954          ASTRING.TAG := 3;
955          ASTRING.CVALUE := II;
956          SEARCH_FIX_TAB(IL,ASTRING);
957          IF (IL = FIX_COUNTER) THEN BEGIN
958             FIX_TAB[IL].TAG := ASTRING.TAG;
959             FIX_TAB[IL].CVALUE := ASTRING.CVALUE;
960             FIX_COUNTER:=SUCC(FIX_COUNTER)
961             END "IF";
962          AUNIT.CLAS := 'CONST  ';
963          AUNIT.INDEX := IL
964          END ELSE BEGIN
965          ERROR(CURNT_ST_NO,'STRING_ERROR        ');
966          AUNIT.CLAS := 'CONST  ';
967          AUNIT.INDEX := 0
968          END
969     END;
970
971
972   "*************************************************************************
973   *  GET_TOK(VAR UNIT:ATOK;MARK0,MARK1:V_MARK);                            *
974   *    1. (MARK0=1):EXPECT GET PROCEDURE ID,(MARK=0):EXPECT GET ID.        *
975   *    2. (MARK1=1):DEFINE IDENTIFIER,(MARK1=0):REFERENCE IDENTIFIER       *
976   *    3. CALL IDENTIFIER,NUMBER,STRING.                                   *
977   *************************************************************************"
978
979   PROCEDURE GET_TOK"(VAR UNIT:ATOK; MARK0,MARK1:V_MARK;VAR HHKY:INTEGER)";
980     BEGIN
981       WHILE (CH IN [' ',NL,'"']) DO
982         BEGIN
983           IF (CH = '"') THEN BEGIN
984              REPEAT
985                WRITE(CH); READ(CH)
```

```
986               UNTIL (CH = '"');
987               WRITE(CH); READ(CH)
988               END ELSE REPEAT
989               WRITE(CH); READ(CH)
990               UNTIL NOT(CH IN [' ',NL])
991            END;
992         CASE CH OF
993              'A','B','C','D','E','F','G','H',
994              'I','J','K','L','M','N','O','P',
995              'Q','R','S','T','U','V','W','X',
996              'Y','Z','_','a','b','c','d','e',
997              'f','g','h','i','j','k','l','m',
998              'n','o','p','q','r','s','t','u',
999              'v','w','x','y','z'              : IDENTIFIER(MARKO,MARK1,HHKY);
1000             ',','.','(',')','[',']','&',';': BEGIN
1001                                                UNIT.CLAS := '       ';
1002                                                UNIT.CLAS[0] := CH;
1003                                                UNIT.INDEX    := 0;
1004                                                WRITE(CH); READ(CH)
1005                                               END;
1006             '1','2','3','4','5','6','7','8','9',
1007             '0'                                : NUMBER;
1008             ':' : BEGIN
1009                     WRITE(CH); READ(CH);
1010                     IF (CH = '=') THEN BEGIN
1011                        UNIT.CLAS := 'OP     ';
1012                        UNIT.INDEX := 6;
1013                        WRITE(CH); READ(CH)
1014                        END ELSE BEGIN
1015                        UNIT.CLAS := ':      ';
1016                        UNIT.INDEX := 0
1017                        END
1018                   END;
1019             '''' : STRING;
1020             '+' : BEGIN
1021                     UNIT.CLAS := 'OP     ';
1022                     UNIT.INDEX := 1;
1023                     WRITE(CH); READ(CH)
1024                   END;
1025             '-' : BEGIN
1026                     UNIT.CLAS := 'OP     ';
1027                     UNIT.INDEX := 2;
1028                     WRITE(CH);
1029                     READ(CH)
1030                   END;
1031             '/' : BEGIN
1032                     UNIT.CLAS := 'OP     ';
1033                     UNIT.INDEX := 3;
1034                     WRITE(CH); READ(CH)
1035                   END;
1036             '*' : BEGIN
1037                     WRITE(CH); READ(CH);
1038                     IF (CH = '*') THEN BEGIN
1039                        UNIT.CLAS := 'OP     ';
1040                        UNIT.INDEX := 5;
1041                        WRITE(CH); READ(CH)
1042                        END ELSE BEGIN
1043                        UNIT.CLAS := 'OP     ';
```

```
1044                        UNIT.INDEX := 4
1045                        END
1046                    END;
1047            '>' : BEGIN
1048                    WRITE(CH); READ(CH);
1049                    IF (CH = '=') THEN BEGIN
1050                       UNIT.CLAS := 'COMP_OP';
1051                       UNIT.INDEX := 3;
1052                       WRITE(CH); READ(CH)
1053                       END ELSE BEGIN
1054                       UNIT.CLAS := 'COMP_OP';
1055                       UNIT.INDEX := 2
1056                       END
1057                    END;
1058            '=' : BEGIN
1059                    UNIT.CLAS := 'COMP_OP';
1060                    UNIT.INDEX := 1;
1061                    WRITE(CH);
1062                    READ(CH)
1063                    END;
1064            '<' : BEGIN
1065                    WRITE(CH); READ(CH);
1066                    IF (CH = '=') THEN BEGIN
1067                       UNIT.CLAS := 'COMP_OP';
1068                       UNIT.INDEX := 5;
1069                       WRITE(CH); READ(CH)
1070                       END ELSE
1071                       IF (CH = '>') THEN BEGIN
1072                          UNIT.CLAS := 'COMP_OP';
1073                          UNIT.INDEX := 6;
1074                          WRITE(CH); READ(CH)
1075                          END ELSE BEGIN
1076                          UNIT.CLAS := 'COMP_OP';
1077                          UNIT.INDEX := 4
1078                          END
1079                    END;
1080            EM : BEGIN
1081                   UNIT.CLAS := '       ';
1082                    UNIT.CLAS[0] := EM;
1083                    UNIT.INDEX := 0
1084                    END;
1085            ELSE : BEGIN
1086                     UNIT.CLAS := '???????';
1087                     UNIT.INDEX := 0;
1088                     WRITE(CH);
1089                     READ(CH)
1090                    END
1091       END "CASE"
1092     END;
1093
1094
1095   "****************** SYNC:SYNC_TABLE ************************************"
1096   PROCEDURE SYNC(A:SYNC_TABLE; K:INTEGER);
1097     VAR GETIT:BOOLEAN; I:INTEGER;
1098     BEGIN
1099       GETIT:=FALSE;
1100       WHILE NOT(GETIT) DO BEGIN
1101         I:=0;
```

```
1102          REPEAT
1103            I:=I + 1;
1104            IF (AUNIT.CLAS=A[I]) THEN GETIT:=TRUE
1105          UNTIL (GETIT) OR (I=K);
1106          IF (GETIT=FALSE) THEN GET_TOK(AUNIT,0,0,AHKY)
1107        END "WHILE"
1108      END;
1109
1110
1111    "*************************************************************************
1112    *  STIMU(VAR ERR_VALUE:BOOLEAN;VAR CHSIDX:CINDEX);                      *
1113    *    1.IT COVERS SEVERAL INTERNAL PROCEDURES:CHECK_TYPE,EXPR,SEXPR,     *
1114    *      TERM,FACTOR,SEARCH_LIST                                          *
1115    *    2.EXPR,SEXPR,TERM CALL CHECK_TYPE FOR TYPE_CHECKING AND RETURN     *
1116    *      THE OPERNAD'S TYPE                                               *
1117    *    3.STIMU_TAB:ARRAY[1..999] OF CHAR                                  *
1118    *    4.CHSIDX:THE POSITION OF THE CURRENT STIMULATION IN STIMU_TAB      *
1119    *    5.INSERT THE CURRENT STIMULATION TO THE STIMU_TAB, AT THE END      *
1120    *      OF THIS STIMULATION INSERT '$' TO THE STIMU_TAB                  *
1121    *************************************************************************"
1122
1123    PROCEDURE STIMU(VAR ERR_VALUE:BOOLEAN;VAR CHSIDX:CINDEX);
1124    VAR I,J,K:INTEGER; SEARCHIT:BOOLEAN; OP_STORE:ID_TYPE;
1125        EXPR_TYPE,TERM_TYPE,FACTOR_TYPE:VALUETYPE; ANO:CINDEX;
1126        CURNT_SEXPR_TYPE,CURNT_TERM_TYPE:VALUETYPE;
1127        CURNT_FACTOR_TYPE:VALUETYPE; COND1_CODE:CODE_TYPE;
1128        MK0,MK1:V_MARK;
1129
1130    "*** CHECK_TYPE:BE CALLED TO DO TYPE_CHECKING & RETURN OPRND_TYPE ***"
1131        PROCEDURE CHECK_TYPE(VAR OPRND_TYPE:VALUETYPE;
1132                             CURNT_OPRND_TYPE:VALUETYPE);
1133        BEGIN
1134          IF(OP_STORE='OP      ') THEN
1135            IF(OPRND_TYPE<>CURNT_OPRND_TYPE) THEN
1136              IF((OPRND_TYPE=1)&(CURNT_OPRND_TYPE=2))
1137                OR((OPRND_TYPE=2)&(CURNT_OPRND_TYPE=1))
1138                THEN BEGIN
1139                OPRND_TYPE := 2;
1140                ERROR(CURNT_ST_NO,'WARNING:CONV INTEGER')
1141                END ELSE BEGIN
1142                OPRND_TYPE := 1;
1143                ERROR(CURNT_ST_NO,'OPRND TYPE CONFLICT ')
1144                END ELSE
1145              IF(OPRND_TYPE=3)OR(OPRND_TYPE=4) THEN BEGIN
1146                OPRND_TYPE := 1;
1147                ERROR(CURNT_ST_NO,'OPRND TYPE CONFLICT ')
1148                END;
1149          IF((OP_STORE='AND     ')OR(OP_STORE='OR      ')) &
1150            ((OPRND_TYPE<>4)&(CURNT_OPRND_TYPE<>4)) THEN BEGIN
1151            OPRND_TYPE := 1;
1152            ERROR(CURNT_ST_NO,'OPRND TYPE CONFLICT ')
1153            END;
1154          IF(OP_STORE='COMP_OP') THEN
1155            IF(OPRND_TYPE<>CURNT_OPRND_TYPE) THEN
1156              IF((OPRND_TYPE=1)&(CURNT_OPRND_TYPE=2)) OR
1157                ((OPRND_TYPE=2)&(CURNT_OPRND_TYPE=1))
1158                THEN BEGIN
1159                OPRND_TYPE := 4;
```

```
1160                ERROR(CURNT_ST_NO,'WARNING:CONV INTEGER')
1161                END ELSE BEGIN
1162                OPRND_TYPE := 1;
1163                ERROR(CURNT_ST_NO,'OPRND TYPE CONFLICT ')
1164                END ELSE
1165              IF(OPRND_TYPE=4) THEN BEGIN
1166                OPRND_TYPE := 1;
1167                ERROR(CURNT_ST_NO,'OPRND TYPE CONFLICT ')
1168                END ELSE
1169                OPRND_TYPE := 4
1170        END;
1171
1172
1173    "****** FACTOR:COLLECT FACTOR PART AND CALLED EXPR & FACTOR ITSELF ***"
1174        PROCEDURE EXPR; FORWARD;
1175        PROCEDURE FACTOR; FORWARD;
1176        PROCEDURE FACTOR;
1177        BEGIN
1178         MK0:=0; MK1:=0;
1179         IF (AUNIT.CLAS='CONST  ') THEN BEGIN
1180            I:=I + 1; COND1_CODE[I]:='F';
1181           I:=I + 1; COND1_CODE[I]:='(';
1182            CONV_NUM(ANO,AUNIT.INDEX);
1183            K:=1;
1184            REPEAT
1185              I:=I + 1;
1186              COND1_CODE[I]:=ANO[K]; K:=K+1
1187            UNTIL (ANO[K]=' ') OR (K=4);
1188           I:=I + 1; COND1_CODE[I]:=')';
1189            CURNT_FACTOR_TYPE:=FIX_TAB[AUNIT.INDEX].TAG;
1190            GET_TOK(AUNIT,MK0,MK1,AHKY)
1191            END ELSE
1192         IF (AUNIT.CLAS='ID     ') THEN BEGIN
1193           I:=I + 1; COND1_CODE[I]:='D';
1194           I:=I + 1; COND1_CODE[I]:='(';
1195            CONV_NUM(ANO,ID_TAB[AUNIT.INDEX].DIDX);
1196            K:=1;
1197            REPEAT
1198              I:=I + 1;
1199              COND1_CODE[I]:=ANO[K]; K:=K+1
1200            UNTIL (ANO[K]=' ') OR (K=4);
1201           I:=I + 1; COND1_CODE[I]:=')';
1202            CURNT_FACTOR_TYPE:=
1203            DICT_TAB[ID_TAB[AUNIT.INDEX].DIDX].ATTRIBUTE;
1204            GET_TOK(AUNIT,MK0,MK1,AHKY)
1205            END ELSE
1206         IF (AUNIT.CLAS='NOT    ') THEN BEGIN
1207            I:=I + 1; COND1_CODE[I]:='N';
1208            I:=I + 1; COND1_CODE[I]:='O';
1209            I:=I + 1; COND1_CODE[I]:='T';
1210            I:=I + 1; COND1_CODE[I]:='(';
1211            GET_TOK(AUNIT,MK0,MK1,AHKY);
1212            FACTOR;
1213            I:=I + 1; COND1_CODE[I]:=')'
1214            END ELSE
1215         IF (AUNIT.CLAS='(      ') THEN BEGIN
1216            I:=I + 1; COND1_CODE[I]:='(';
1217            GET_TOK(AUNIT,MK0,MK1,AHKY);
```

```
1218            EXPR;
1219            CURNT_FACTOR_TYPE:=EXPR_TYPE;
1220            IF (AUNIT.CLAS=')       ') THEN BEGIN
1221               I:=I + 1; COND1_CODE[I]:=')';
1222               GET_TOK(AUNIT,MK0,MK1,AHKY)
1223               END ELSE ERROR(CURNT_ST_NO,'PARENTHES NOT MATCH ')
1224            END ELSE
1225         IF (AUNIT.CLAS='ABS     ') THEN BEGIN
1226            K:=0;
1227            REPEAT
1228              I:=I+1; COND1_CODE[I]:=AUNIT.CLAS[K]; K:=K+1
1229            UNTIL (AUNIT.CLAS[K]=' ');
1230            GET_TOK(AUNIT,MK0,MK1,AHKY);
1231            IF (AUNIT.CLAS='(       ') THEN BEGIN
1232               I:=I+1; COND1_CODE[I]:='('
1233               END ELSE
1234               ERROR(CURNT_ST_NO,'FACTOR SYNTAX ERROR ');
1235            GET_TOK(AUNIT,MK0,MK1,AHKY);
1236            EXPR;
1237            IF (AUNIT.CLAS=')       ') THEN BEGIN
1238               I:=I+1; COND1_CODE[I]:=')'
1239               END ELSE
1240               ERROR(CURNT_ST_NO,'PARENTHES NOT MATCH ');
1241            CURNT_FACTOR_TYPE:=4
1242            END ELSE
1243         IF (AUNIT.CLAS='V_EXIST') THEN BEGIN
1244            K:=0;
1245            REPEAT
1246              I:=I+1; COND1_CODE[I]:=AUNIT.CLAS[K]; K:=K+1
1247            UNTIL (K=7);
1248            GET_TOK(AUNIT,MK0,MK1,AHKY);
1249            IF (AUNIT.CLAS='(       ') THEN BEGIN
1250               I:=I+1; COND1_CODE[I]:='('
1251               END ELSE
1252               ERROR(CURNT_ST_NO,'FACTOR SYNTAX ERROR ');
1253            GET_TOK(AUNIT,MK0,MK1,AHKY);
1254            IF (AUNIT.CLAS='ID      ') THEN BEGIN
1255               I:=I+1; COND1_CODE[I]:='D';
1256               CONV_NUM(ANO,ID_TAB[AUNIT.INDEX].DIDX);
1257               K:=1;
1258               REPEAT
1259                 I:=I+1; COND1_CODE[I]:=ANO[K]; K:=K+1
1260               UNTIL (ANO[K]=' ') OR (K=5)
1261               END ELSE
1262               ERROR(CURNT_ST_NO,'V_EXIST SYNTAX ERROR');
1263            GET_TOK(AUNIT,MK0,MK1,AHKY);
1264            IF (AUNIT.CLAS=')       ') THEN BEGIN
1265               I:=I+1; COND1_CODE[I]:=')'
1266               END ELSE
1267               ERROR(CURNT_ST_NO,'PARENTHES NOT MATCH ');
1268            CURNT_FACTOR_TYPE:=4
1269            END ELSE BEGIN
1270            ERROR(CURNT_ST_NO,'EXPR FACTOR ERROR    ');
1271            CURNT_FACTOR_TYPE:=1
1272            END
1273      END;
1274
1275
```

```
1276    "********** TERM:COLLECT TERM PART AND CALL CHECK_TYPE **************"
1277       PROCEDURE TERM;
1278       VAR OUTERM:BOOLEAN;
1279       BEGIN
1280         MK0:=0; MK1:=0; OUTERM:=FALSE;
1281         FACTOR;
1282         FACTOR_TYPE:=CURNT_FACTOR_TYPE;
1283         REPEAT
1284           IF (AUNIT.CLAS='OP     ') &
1285              ((AUNIT.INDEX=3)OR(AUNIT.INDEX=4)) THEN BEGIN
1286              I:=I + 1;
1287              COND1_CODE[I]:=OP_TAB[AUNIT.INDEX,1];
1288              OP_STORE:='OP     ';
1289              GET_TOK(AUNIT,MK0,MK1,AHKY)
1290              END ELSE
1291           IF (AUNIT.CLAS='AND    ') THEN BEGIN
1292              I:=I + 1; COND1_CODE[I]:='A';
1293              I:=I + 1; COND1_CODE[I]:='N';
1294              I:=I + 1; COND1_CODE[I]:='D';
1295              OP_STORE:='AND    ';
1296              GET_TOK(AUNIT,MK0,MK1,AHKY)
1297              END ELSE OUTERM:=TRUE;
1298           IF (OUTERM=FALSE) THEN BEGIN
1299              FACTOR;
1300              CHECK_TYPE(FACTOR_TYPE,CURNT_FACTOR_TYPE)
1301             END
1302         UNTIL (OUTERM=TRUE);
1303         CURNT_TERM_TYPE:=FACTOR_TYPE
1304       END;
1305
1306
1307    "******* SEXPR:COLLECT SEXPR PART AND CALL TERM,CHECK_TYPE ***********"
1308       PROCEDURE SEXPR;
1309       VAR OUTSEXPR:BOOLEAN;
1310       BEGIN
1311         MK0:=0; MK1:=0;
1312         IF (AUNIT.CLAS='OP     ')&((AUNIT.INDEX=1)OR(AUNIT.INDEX = 2))
1313            THEN BEGIN
1314            I:=I + 1;
1315            COND1_CODE[I]:=OP_TAB[AUNIT.INDEX,1];
1316            GET_TOK(AUNIT,MK0,MK1,AHKY)
1317            END;
1318         TERM;
1319         TERM_TYPE:=CURNT_TERM_TYPE;
1320         REPEAT
1321           IF (AUNIT.CLAS='OP     ') &
1322              ((AUNIT.INDEX=1) OR (AUNIT.INDEX=2))
1323              THEN BEGIN
1324              I:=I + 1;
1325              COND1_CODE[I]:=OP_TAB[AUNIT.INDEX,1];
1326              OP_STORE:='OP     ';
1327              GET_TOK(AUNIT,MK0,MK1,AHKY)
1328              END ELSE
1329           IF (AUNIT.CLAS='OR     ') THEN BEGIN
1330              I:=I + 1; COND1_CODE[I]:='O';
1331              I:=I + 1; COND1_CODE[I]:='R';
1332              OP_STORE:='OR     ';
1333              GET_TOK(AUNIT,MK0,MK1,AHKY)
```

```
1334            END ELSE OUTSEXPR:=TRUE;
1335          IF (OUTSEXPR=FALSE) THEN BEGIN
1336            TERM;
1337            CHECK_TYPE(TERM_TYPE,CURNT_TERM_TYPE)
1338            END
1339        UNTIL (OUTSEXPR=TRUE);
1340        CURNT_SEXPR_TYPE:=TERM_TYPE
1341      END;
1342
1343
1344    "********** EXPR:CONSTRUCT EXPR AND CALL SEXPR,CHECK_TYPE ************"
1345      PROCEDURE EXPR;
1346      BEGIN
1347        MK0:=0; MK1:=0;
1348        SEXPR;
1349        IF (AUNIT.CLAS='COMP_OP') THEN BEGIN
1350           K:=1;
1351           REPEAT
1352             I:=I + 1;
1353             COND1_CODE[I]:=COMP_OP_TAB[AUNIT.INDEX,K];
1354             K:=K+1
1355           UNTIL (K=3) OR (COMP_OP_TAB[AUNIT.INDEX,K]=' ');
1356           OP_STORE:='COMP_OP';
1357           GET_TOK(AUNIT,MK0,MK1,AHKY);
1358           EXPR_TYPE:=CURNT_SEXPR_TYPE;
1359           SEXPR;
1360           CHECK_TYPE(EXPR_TYPE,CURNT_SEXPR_TYPE)
1361           END
1362      END;
1363
1364      "*BEGIN*"
1365      BEGIN
1366         FOR I:=1 TO 70 DO COND1_CODE[I]:=' ';
1367         I:=0; MK0:=0; MK1:=0;
1368         GET_TOK(AUNIT,MK0,MK1,AHKY);
1369         EXPR;
1370         IF (AUNIT.CLAS=']       ')
1371            THEN GET_TOK(AUNIT,1,MK1,AHKY)
1372            ELSE ERROR(CURNT_ST_NO,'STIMU SYNTAX ERROR  ');
1373         CONV_NUM(CHSIDX,STIMU_IDX);
1374         K:=1;
1375         REPEAT
1376           STIMU_TAB[STIMU_IDX]:=COND1_CODE[K];
1377           STIMU_IDX:=SUCC(STIMU_IDX);
1378           K:=K + 1
1379         UNTIL (COND1_CODE[K]=' ') OR (K=71);
1380         STIMU_TAB[STIMU_IDX]:=';';
1381         STIMU_IDX:=SUCC(STIMU_IDX)
1382      END;
1383
1384
1385    "*****************************************************************************
1386    *  PARM_ARG:                                                                 *
1387    *    1.THE PARAMETERS:PID_PT:THE INDEX OF CURRENT PID IN PID_TAB             *
1388    *                       AOP:THE ACTION NAME OF CURRENT REQUEST               *
1389    *                       (PA=0):PARAMETERS,(PA=1):ARGUMENTS                   *
1390    *    2.CALL MR_PROC(MR:V_MARK);                                              *
1391    *      01.(MR=0):OUTPUT PART,(MR=1):INPUT PART                               *
```

```
1392    *      02.INSERT PID_TAB[PID_PT].OUPUTL(AOUPUTL)&INPUTL(AINPUTL)      *
1393    *    3.INSERT PID_TAB[PID_PT].OLEN(AOLEN)&ILEN(AILEN)                 *
1394    *    4.RETURN INPUTLIST,OUPUTLIST:ARRAY[1..25] OF CHAR                *
1395    *      INPUTLIST:D15,F10,D35.+1,D600.-1,.....                         *
1396    *      OUPUTLIST:25,100.+1,30.-1,200.+0,.....                         *
1397    *    5.CALL SEMAN_CK TO DO SEMANTIC CHECKING                          *
1398    ***********************************************************************"
1399
1400    PROCEDURE PARM_ARG(PID_PT:INTEGER;PA:V_MARK;AOP:TWOCH;
1401                       VAR ERR_MARK:BOOLEAN;VAR CHR,CHM:CLIST);
1402    VAR I,II,J,K,L,RLEN,MLEN,AILEN,AOLEN:INTEGER;
1403        A:CHAR; AA:ID_TYPE; CHIDX:CINDEX;
1404        AOUPUTL:PARMLIST;
1405        AINPUTL:ARRAY[1..5] OF INPARM;
1406
1407
1408    "******************************** MR_PROC ******************************"
1409        PROCEDURE MR_PROC(MR:V_MARK);
1410        VAR KK:INTEGER;
1411        BEGIN
1412          MK0:=0;
1413          IF (PA=1) THEN MK1:=0 ELSE MK1:=1;
1414          I:=0; J:=0; K:=0; L:=0; MLEN:=0; RLEN:=0;
1415          GET_TOK(AUNIT,MK0,MK1,AHKY);
1416          REPEAT
1417            IF (MR=0) THEN RLEN:=RLEN+1 ELSE MLEN:=MLEN+1;
1418            IF (AUNIT.CLAS='ID     ') THEN BEGIN
1419               I:=I + 1;
1420               IF (MR=0) THEN
1421                  "*OUTPUT PART OF PARAMETER OR ARGUMENT*"
1422                  IF (PA=0) THEN
1423                     PID_TAB[PID_PT].OUPUTL[I]:=ID_TAB[AUNIT.INDEX].DIDX
1424                     ELSE AOUPUTL[I]:=ID_TAB[AUNIT.INDEX].DIDX
1425                  "*INPUT PART:ARRAY[1..5] OF INPARM,(FLAG='D'):DICT_TAB*"
1426                  "*(FLAG='F'):FIX_TAB                                  *"
1427                  ELSE BEGIN
1428                  IF (PA=0) THEN BEGIN
1429                     PID_TAB[PID_PT].INPUTL[I].FLAG:='D';
1430                     PID_TAB[PID_PT].INPUTL[I].INDX:=
1431                                      ID_TAB[AUNIT.INDEX].DIDX
1432                     END ELSE BEGIN
1433                     AINPUTL[I].FLAG:='D';
1434                     AINPUTL[I].INDX:=ID_TAB[AUNIT.INDEX].DIDX
1435                     END;
1436                  J:=J+1; CHM[J]:='D'
1437                  END;
1438               CONV_NUM(CHIDX,ID_TAB[AUNIT.INDEX].DIDX);
1439               L:=1;
1440               REPEAT
1441                 J:=J+1;
1442                 IF(MR=0) THEN CHR[J]:=CHIDX[L] ELSE CHM[J]:=CHIDX[L];
1443                 L:=L+1
1444               UNTIL (CHIDX[L]=' ') OR (L=4);
1445               IF (PA=1) THEN MK1:=0 ELSE MK1:=1;
1446               GET_TOK(AUNIT,MK0,MK1,AHKY);
1447               "** HANDLE SOME KIND OF PARAMETER OR ARGUMENT:A.1,A.0,. **"
1448               "** ..,A.+0,A.+1,A.-1,...                              **"
1449               IF (AUNIT.CLAS='.      ') THEN BEGIN
```

```
1450                J:=J + 1;
1451                IF(MR=0) THEN CHR[J]:='.' ELSE CHM[J]:='.';
1452                IF (CH='+') OR (CH='-') THEN BEGIN
1453                   J:=J+1;
1454                   IF (MR=0) THEN CHR[J]:=CH ELSE CHM[J]:=CH;
1455                   WRITE(CH); READ(CH)
1456                   END;
1457                IF (CH IN DIGITS) THEN
1458                   REPEAT
1459                     J:=J+1;
1460                     IF (MR=0) THEN CHR[J]:=CH ELSE CHM[J]:=CH;
1461                     WRITE(CH); READ(CH)
1462                   UNTIL NOT (CH IN DIGITS)
1463                   ELSE ERROR(CURNT_ST_NO,'WRONG PARM_ARG      ');
1464                IF (PA=1) THEN MK1:=0 ELSE MK1:=1;
1465                GET_TOK(AUNIT,MK0,MK1,AHKY)
1466                END
1467             END
1468             ELSE IF (MR=0)
1469                     THEN ERR_MARK:=TRUE
1470                     "*HANDLE CONST PARAMETER OR ARGUMENT*"
1471                     ELSE IF(AUNIT.CLAS='CONST  ') THEN BEGIN
1472                            J:=J+1; CHM[J]:='F'; I:=I+1;
1473                            IF(PA=0) THEN BEGIN
1474                               PID_TAB[PID_PT].INPUTL[I].FLAG:='F';
1475                               PID_TAB[PID_PT].INPUTL[I].INDX:=
1476                               AUNIT.INDEX
1477                               END ELSE BEGIN
1478                               AINPUTL[I].FLAG:='F';
1479                               AINPUTL[I].INDX:=AUNIT.INDEX
1480                               END;
1481                            CONV_NUM(CHIDX,AUNIT.INDEX);
1482                            L:=1;
1483                            REPEAT
1484                              J:=J+1; CHM[J]:=CHIDX[L]; L:=L+1
1485                            UNTIL(CHIDX[L]=' ') OR (L=4);
1486                            GET_TOK(AUNIT,MK0,MK1,AHKY)
1487                            END
1488                            ELSE ERR_MARK:=TRUE;
1489       "FOR KK:=1 TO 5 DO WRITE('*'); WRITE(' ');
1490       FOR KK:=1 TO 25 DO WRITE(CHR[KK]);
1491       FOR KK:=1 TO 5 DO WRITE(CHM[KK]); WRITE(NL);"
1492           IF (MR=0) THEN AA:=':      ' ELSE AA:=')      ';
1493           IF (ERR_MARK=TRUE) THEN BEGIN
1494              IF (MR=0) THEN ERROR(CURNT_ST_NO,'WRONG OUPUT PARM_ARG')
1495                        ELSE ERROR(CURNT_ST_NO,'WRONG INPUT PARM_ARG');
1496              FOR II:=1 TO 10 DO SYNC_TAB[II]:=BLANK;
1497              SYNC_TAB[1] := ',      '; SYNC_TAB[2] := 'ID     ';
1498              SYNC_TAB[3] := 'CONST  '; SYNC_TAB[4] := ':      ';
1499              SYNC_TAB[5] := ')      '; SYNC(SYNC_TAB,5)
1500              END;
1501           IF (AUNIT.CLAS=',      ') THEN BEGIN
1502              J := J + 1;
1503              IF (MR=0) THEN CHR[J]:=',' ELSE CHM[J]:=',';
1504              GET_TOK(AUNIT,MK0,MK1,AHKY)
1505              END ELSE
1506              IF(AUNIT.CLAS<>AA) THEN BEGIN
1507                ERROR(CURNT_ST_NO,'PARM_ARGS SYNTAX_ERR');
```

```
1508                WHILE (AUNIT.CLAS<>';      ') DO
1509                   REPEAT
1510                     GET_TOK(AUNIT,MK0,MK1,AHKY)
1511                   UNTIL (AUNIT.CLAS=';      ')
1512                END
1513          UNTIL (AUNIT.CLAS=AA) OR (AUNIT.CLAS=';      ') OR (I=5)
1514       END;
1515
1516
1517    "*************************************************************************"
1518    *  SEMAN_CK:CHECK THREE THINGS                                            *
1519    *    1.INPUT PART AND OUTPUT PART ARE LEGAL OR NOT                        *
1520    *    2.PARAMETERS AND ARGUMENTS ARE MATCH OR NOT                          *
1521    *    3.OPERAND TYPE CHECKING                                              *
1522    *************************************************************************"
1523
1524       PROCEDURE SEMAN_CK;
1525       VAR PP:PID_TAB_ENTRY; PD,PAO,PI,PAI:DICT_TAB_ENTRY;
1526           ATT:ARRAY[1..3] OF VALUETYPE; SM_ERR:BOOLEAN;
1527           STOPIT:BOOLEAN;
1528       BEGIN
1529         IF (PA<>0) & (PID_PT<>0) THEN BEGIN
1530            PP:=PID_TAB[PID_PT];
1531            IF (PP.OLEN<>AOLEN) THEN BEGIN
1532               SM_ERR := TRUE;
1533               ERROR(CURNT_ST_NO,'OUTARGS_NO<>PARMS_NO')
1534               END ELSE BEGIN
1535
1536               "*CHECK OUPUT PARM_NO AND OUPUT ARGU_NO EQUAL OR NOT*"
1537               "*CHECK OUPUT PARM_TY AND OUPUT ARGU_TY MATCH OR NOT*"
1538               I := 0;
1539               REPEAT
1540                 I := I + 1;
1541                 IF (AOUPUTL[I]=0) OR (PP.OUPUTL[I]=0) THEN
1542                    STOPIT:=TRUE ELSE BEGIN
1543                    PAO:=DICT_TAB[AOUPUTL[I]];
1544                    PD:=DICT_TAB[PP.OUPUTL[I]];
1545                    IF (PAO.LONGEVITY<>PD.LONGEVITY) OR
1546                       (PAO.ATTRIBUTE<>PD.ATTRIBUTE) THEN BEGIN
1547                       SM_ERR := TRUE;
1548                       ERROR(CURNT_ST_NO,'OUTARGS_TP<>PARMS_TP')
1549                       END
1550                    END
1551               UNTIL (STOPIT=TRUE) OR (SM_ERR = TRUE)
1552               END;
1553
1554            "*CHECK INPUT PARM_NO AND ARGU_NO EQUAL OR NOT*"
1555            "*CHECK INPUT PARM_TY AND ARGU_TY MATCH OR NOT*"
1556            IF (PP.ILEN<>AILEN) THEN BEGIN
1557               SM_ERR := TRUE;
1558               ERROR(CURNT_ST_NO,'INARGS_NO<>PARMS_NO ')
1559               END ELSE BEGIN
1560               I := 0;
1561               REPEAT
1562                 I := I + 1;
1563                 IF (PP.INPUTL[I].INDX=0) OR (AINPUTL[I].INDX=0) THEN
1564                    STOPIT:=TRUE ELSE BEGIN
1565                    PI:=DICT_TAB[PP.INPUTL[I].INDX];
```

```
1566                    PAI:=DICT_TAB[AINPUTL[I].INDX];
1567                  . IF(PP.INPUTL[I].FLAG='D')&(AINPUTL[I].FLAG='D') THEN
1568                      IF(PI.ATTRIBUTE<>PAI.ATTRIBUTE) OR
1569                        (PI.LONGEVITY<>PAI.LONGEVITY) THEN BEGIN
1570                        SM_ERR := TRUE;
1571                        ERROR(CURNT_ST_NO,'INARGS_TP<>PARMS_TP ')
1572                        END;
1573                    IF(PP.INPUTL[I].FLAG='F')&(AINPUTL[I].FLAG='F') THEN
1574                      IF(PI.ATTRIBUTE<>PAI.ATTRIBUTE) THEN BEGIN
1575                        SM_ERR := TRUE;
1576                        ERROR(CURNT_ST_NO,'INARGS_TP<>PARMS_TP ')
1577                        END;
1578                    IF(PP.INPUTL[I].FLAG='D')&(AINPUTL[I].FLAG='F') THEN
1579                      IF(PI.LONGEVITY<>1) OR
1580                        (PI.ATTRIBUTE<>PAI.ATTRIBUTE) THEN BEGIN
1581                        SM_ERR := TRUE;
1582                        ERROR(CURNT_ST_NO,'INARGS_TP<>PARMS_TP ')
1583                        END
1584                  END
1585              UNTIL (STOPIT=TRUE) OR (SM_ERR=TRUE)
1586              END
1587          END;
1588
1589      "*CHECH AN ACTION'S INPUT PART AND OUPUT PART ARE ILLEGAL*"
1590      "*OR LEGAL,REASONABLE OR UNREASONABLE                    *"
1591      IF (PID_PT=0) THEN
1592         IF(AOP='+ ')OR(AOP='- ')OR(AOP='* ')OR(AOP='/ ') THEN
1593           IF (AILEN=2) & (AOLEN=1) THEN BEGIN
1594              IF (AINPUTL[I].FLAG='D') THEN
1595                 ATT[1]:=DICT_TAB[AINPUTL[1].INDX].ATTRIBUTE
1596                 ELSE ATT[1]:=FIX_TAB[AINPUTL[1].INDX].TAG;
1597              IF (AINPUTL[2].FLAG='F') THEN
1598                 ATT[2]:=DICT_TAB[AINPUTL[2].INDX].ATTRIBUTE
1599                 ELSE ATT[2]:=FIX_TAB[AINPUTL[2].INDX].TAG;
1600              ATT[3]:=DICT_TAB[AOUPUTL[1]].ATTRIBUTE;
1601              IF(ATT[1]>2)OR(ATT[2]>2)OR(ATT[3]>2) THEN BEGIN
1602                SM_ERR := TRUE;
1603                ERROR(CURNT_ST_NO,'OPRND TYPE CONFLICT ')
1604                END ELSE
1605                IF(ATT[1]<>ATT[2])OR(ATT[2]<>ATT[3])OR
1606                  (ATT[1]<>ATT[3]) THEN
1607                  ERROR(CURNT_ST_NO,'WARNING:CONV INTEGER');
1608              END ELSE BEGIN
1609              SM_ERR := TRUE;
1610              ERROR(CURNT_ST_NO,'ILLEGAL IN_OUPUT_NO ')
1611              END;
1612         IF (AOP=':=') THEN
1613            IF (AILEN<>1) & (AILEN<>AOLEN) THEN BEGIN
1614               SM_ERR := TRUE;
1615               ERROR(CURNT_ST_NO,'ILLEGAL IN_OUPUT_NO ')
1616               END ELSE BEGIN
1617               IF (AILEN=1) THEN BEGIN
1618                  IF(AINPUTL[1].FLAG='D')
1619                    THEN ATT[1]:=DICT_TAB[AINPUTL[1].INDX].ATTRIBUTE
1620                    ELSE ATT[1]:=FIX_TAB[AINPUTL[1].INDX].TAG;
1621                  I := 0;
1622                  REPEAT
1623                    I := I + 1;
```

```
1624                    IF (AOUPUTL[I]=0) THEN STOPIT:=TRUE ELSE
1625                    IF(DICT_TAB[AOUPUTL[I]].ATTRIBUTE<>ATT[1])
1626                      THEN BEGIN
1627                      SM_ERR := TRUE;
1628                      ERROR(CURNT_ST_NO,'OPRND TYPE CONFLICT ')
1629                      END
1630                  UNTIL (STOPIT=TRUE)  OR (SM_ERR=TRUE) OR (I=5)
1631                  END;
1632               IF (AILEN=AOLEN) THEN BEGIN
1633                  I := 0;
1634                  REPEAT
1635                    I := I + 1;
1636                    IF (AINPUTL[I].INDX=0) OR (AOUPUTL[I]=0) THEN
1637                       STOPIT:=TRUE ELSE BEGIN
1638                       IF(AINPUTL[I].FLAG='F') THEN
1639                         IF(DICT_TAB[AOUPUTL[I]].ATTRIBUTE<>
1640                            FIX_TAB[AINPUTL[I].INDX].TAG)
1641                           THEN BEGIN
1642                           SM_ERR := TRUE;
1643                           ERROR(CURNT_ST_NO,'OPRND TYPE CONFLICT ')
1644                           END;
1645                       IF(AINPUTL[I].FLAG='D') THEN
1646                         IF(DICT_TAB[AOUPUTL[I]].ATTRIBUTE<>
1647                           DICT_TAB[AINPUTL[I].INDX].ATTRIBUTE)
1648                           THEN BEGIN
1649                           SM_ERR := TRUE;
1650                           ERROR(CURNT_ST_NO,'OPRND TYPE CONFLICT ')
1651                           END
1652                         END
1653                  UNTIL (STOPIT=TRUE) OR (SM_ERR=TRUE)
1654                  END
1655               END
1656        END;
1657
1658        "*BEGIN*"
1659        BEGIN
1660          CHM:=BLANK2; CHR:=BLANK2;
1661          FOR I:=1 TO 5 DO AOUPUTL[I]:=0;
1662          FOR I:=1 TO 5 DO BEGIN
1663          AINPUTL[I].INDX:=0; AINPUTL[I].FLAG:=' ' END;
1664          MR_PROC(0);
1665          IF (AUNIT.CLAS<>':      ')
1666             THEN ERROR(CURNT_ST_NO,'MISSING SEMICOLON   ');
1667          IF (PA=0) THEN PID_TAB[PID_PT].OLEN:=RLEN
1668                    ELSE AOLEN:=RLEN;
1669          MR_PROC(1);
1670          IF (AUNIT.CLAS<>')      ')
1671             THEN ERROR(CURNT_ST_NO,'MISSING RPARENTH    ');
1672          IF (PA=0) THEN PID_TAB[PID_PT].ILEN:=MLEN
1673                    ELSE AILEN:=MLEN;
1674          SEMAN_CK;
1675          GET_TOK(AUNIT,1,0,AHKY)
1676        END;
1677
1678
1679    "*************************************************************************
1680    *  TERMI:                                                               *
1681    *    1.IN SOURCE LANGUAGE THERE IS ONLY ONE FORM:INC ID(I,J,K)          *
```

```
1682    *    2.TERMI_TAB:ARRAY[1..30] OF CODE2_TYPE                          *
1683    *    3.INITIALIZE CODE2_CODE=FOR D(   ):=F(   )TO  (   )BY F(   )DO  *
1684    *    4.FILLED OUT CODE2_CODE                                         *
1685    *    5.INSERT CODE2_CODE TO TERMI_TAB                                *
1686    *    6.CHTIDX THE POSITION OF CURRENT TERMILATION IN THE TERMI_TAB   *
1687    ********************************************************************"
1688
1689    PROCEDURE TERMI(VAR ERR_VALUE:BOOLEAN;VAR CHTIDX:CINDEX;PA:V_MARK;
1690                    VAR T_I,T_C:THREE_ARY);
1691    VAR CODE2_CODE:CODE2_TYPE; SM_ERR:BOOLEAN;
1692        ANO:CINDEX; I,K:INTEGER;
1693
1694    "********** ID_CONST_STEP:(SYM=0):IDENTIFIER,(SYM=1):CONST ***********"
1695        PROCEDURE ID_CONST_STEP(SYM:V_MARK;STNO:INTEGER);
1696        BEGIN
1697          IF (PA=1) THEN
1698             IF (DICT_TAB[ID_TAB[AUNIT.INDEX].DIDX].ATTRIBUTE<>1) OR
1699                (FIX_TAB[AUNIT.INDEX].TAG<>1) THEN
1700                ERROR(CURNT_ST_NO,'TERMI TYPE CONFLICT ');
1701             IF(SYM=0) THEN CONV_NUM(ANO,ID_TAB[AUNIT.INDEX].DIDX)
1702                       ELSE CONV_NUM(ANO,AUNIT.INDEX);
1703             K := 1;
1704             I := STNO;
1705             REPEAT
1706               I := I + 1;
1707               CODE2_CODE[I] := ANO[K];
1708               K := K + 1
1709             UNTIL (ANO[K] = ' ') OR (K = 4);
1710             GET_TOK(AUNIT,MK0,MK1,AHKY)
1711        END;
1712
1713        "*BEGIN*"
1714        BEGIN
1715          FOR I:=1 TO 3 DO BEGIN T_I[I]:=0; T_C[I]:=0 END;
1716          MK0:=0; MK1:=1;
1717          CODE2_CODE:='FOR D(   ):=F(   )TO  (   )BY F(   )DO';
1718          GET_TOK(AUNIT,MK0,MK1,AHKY);
1719          IF (AUNIT.CLAS = 'INC    ')
1720             THEN GET_TOK(AUNIT,MK0,MK1,AHKY)
1721             ELSE BEGIN
1722                    ERROR(CURNT_ST_NO,'TERMI MISSING RWORD ');
1723                    FOR I := 1 TO 10 DO SYNC_TAB[I] := BLANK;
1724                    SYNC_TAB[1] := 'ID     ';
1725                    SYNC_TAB[2] := '(      ';
1726                    SYNC_TAB[3] := 'CONST  ';
1727                    SYNC(SYNC_TAB,3)
1728                  END;
1729          IF (AUNIT.CLAS = 'ID     ') THEN BEGIN
1730             T_I[1]:=ID_TAB[AUNIT.INDEX].DIDX ;
1731             ID_CONST_STEP(0,6) END ELSE BEGIN
1732             ERROR(CURNT_ST_NO,'TERMI SYNTAX ERROR  ');
1733             FOR I := 1 TO 10 DO SYNC_TAB[I] := BLANK;
1734             SYNC_TAB[1] := '(      ';
1735             SYNC_TAB[2] := 'CONST  ';
1736             SYNC_TAB[3] := ',      ';
1737             SYNC(SYNC_TAB,3)
1738           END;
1739          IF (AUNIT.CLAS ='(      ')
```

```
1740            THEN GET_TOK(AUNIT,MK0,MK1,AHKY)
1741            ELSE BEGIN
1742                   ERROR(CURNT_ST_NO,'TERMI MISSING "("   ');
1743                   FOR I := 1 TO 10 DO SYNC_TAB[I] := BLANK;
1744                   SYNC_TAB[1] := 'CONST  ';
1745                   SYNC_TAB[2] := ',      ';
1746                   SYNC(SYNC_TAB,2)
1747                 END;
1748          IF (AUNIT.CLAS = 'CONST  ') THEN BEGIN
1749             T_C[1]:=AUNIT.INDEX;
1750             ID_CONST_STEP(1,14) END ELSE BEGIN
1751             ERROR(CURNT_ST_NO,'TERMI SYNTAX ERROR  ');
1752             FOR I := 1 TO 10 DO SYNC_TAB[I] := BLANK;
1753             SYNC_TAB[1] := ',      ';
1754             SYNC_TAB[2] := 'CONST  ';
1755             SYNC_TAB[3] := 'OP     ';
1756             SYNC(SYNC_TAB,3)
1757             END;
1758          IF (AUNIT.CLAS = ',      ')
1759             THEN GET_TOK(AUNIT,MK0,MK1,AHKY)
1760             ELSE BEGIN
1761                   ERROR(CURNT_ST_NO,'TERMI MISSING COMMA ');
1762                   FOR I := 1 TO 10 DO SYNC_TAB[I] := BLANK;
1763                   SYNC_TAB[1] := 'OP     ';
1764                   SYNC_TAB[2] := 'CONST  '
1765                 END;
1766          IF (AUNIT.CLAS = 'OP   ')
1767             THEN BEGIN
1768                   CODE2_CODE[30] := OP_TAB[AUNIT.INDEX,1];
1769                   GET_TOK(AUNIT,MK0,MK1,AHKY)
1770                 END;
1771          IF (AUNIT.CLAS = 'CONST  ') THEN BEGIN
1772             T_C[2]:=AUNIT.INDEX ;
1773             ID_CONST_STEP(1,32) END ELSE BEGIN
1774             ERROR(CURNT_ST_NO,'TERMI SYNTAX ERROR  ');
1775             FOR I := 1 TO 10 DO SYNC_TAB[I] := BLANK;
1776             SYNC_TAB[1] := ',      ';
1777             SYNC_TAB[2] := 'CONST  ';
1778             SYNC_TAB[3] := 'ID     ';
1779             SYNC(SYNC_TAB,3)
1780             END;
1781          IF (AUNIT.CLAS = ',      ')
1782             THEN GET_TOK(AUNIT,MK0,MK1,AHKY)
1783             ELSE BEGIN
1784                   ERROR(CURNT_ST_NO,'TERMI MISSING COMMA ');
1785                   FOR I := 1 TO 10 DO SYNC_TAB[I] := BLANK;
1786                   SYNC_TAB[1] := 'CONST  ';
1787                   SYNC_TAB[2] := 'ID     ';
1788                   SYNC(SYNC_TAB,2)
1789                 END;
1790        IF (AUNIT.CLAS = 'CONST  ') THEN BEGIN
1791           T_C[3]:=AUNIT.INDEX ;
1792           ID_CONST_STEP(1,23); CODE2_CODE[22]:='F'
1793           END ELSE BEGIN
1794           IF (AUNIT.CLAS = 'ID     ') THEN BEGIN
1795              T_I[2]:=ID_TAB[AUNIT.INDEX].DIDX;
1796              CODE2_CODE[22] := 'D';
1797              ID_CONST_STEP(0,23)
```

```
1798              END ELSE ERROR(CURNT_ST_NO,'TERMI SYNTAX ERROR  ')
1799            END;
1800          IF (AUNIT.CLAS = ')      ')
1801             THEN GET_TOK(AUNIT,MK0,MK1,AHKY)
1802             ELSE BEGIN
1803                    ERROR(CURNT_ST_NO,'TERMI MISSING ")"   ');
1804                    FOR I := 1 TO 10 DO SYNC_TAB[I] := BLANK;
1805                    SYNC_TAB[1] := ']      ';
1806                    SYNC_TAB[2] := ';      ';
1807                    SYNC(SYNC_TAB,2)
1808                  END;
1809          IF (AUNIT.CLAS = ']      ')
1810             THEN GET_TOK(AUNIT,MK0,MK1,AHKY)
1811             ELSE BEGIN
1812             ERROR(CURNT_ST_NO,'TERMI SYNTAX ERROR  ');
1813             WHILE (AUNIT.CLAS<>';      ') DO
1814             REPEAT
1815               GET_TOK(AUNIT,1,0,AHKY)
1816             UNTIL (AUNIT.CLAS=';      ')
1817             END;
1818          TERMI_IDX := SUCC(TERMI_IDX);
1819          TERMI_TAB[TERMI_IDX] := CODE2_CODE;
1820          CONV_NUM(CHTIDX,TERMI_IDX)
1821        END;
1822
1823
1824     "*************************************************************************
1825     *  CREATE_CODE:                                                         *
1826     *    1.COLLECT FIVE PARTS:STIMULATIN,ACTIONNAME,TERMINATION,            *
1827     *      INPUT LIST,OUTPUT LIST TO CONSTUCT ONE SMARRT STATEMENT          *
1828     *      INTERMEDIATE CODE                                                *
1829     *    2.CALL ADD_TERM AND ADD_LIST                                       *
1830     *************************************************************************"
1831
1832     PROCEDURE CREATE_CODE(VAR CTEMP:CODE_TYPE;CS,CA,CT:CINDEX;
1833                           CML,CRT:CLIST);
1834       VAR I,J,K : INTEGER;
1835
1836     "****************************** ADD_TERM ******************************"
1837       PROCEDURE ADD_TERM(ATERM:CINDEX;L:INTEGER);
1838         BEGIN
1839           K := 1;
1840           REPEAT
1841             I := I + 1;
1842             CTEMP[I] := ATERM[K];
1843             K := K + 1
1844           UNTIL (K = L) OR (ATERM[K] = ' ')
1845         END;
1846
1847     "****************************** ADD_LIST ******************************"
1848       PROCEDURE ADD_LIST(ALIST:CODE_TYPE;L:INTEGER);
1849         BEGIN
1850           K := 1;
1851           REPEAT
1852             I := I + 1;
1853             CTEMP[I] := ALIST[K];
1854             K := K + 1
1855           UNTIL (K = L) OR (ALIST[K] = ' ')
```

```
1856        END;
1857
1858      "*BEGIN*"
1859      BEGIN
1860        I := 1;
1861        FOR J := 1 TO 70 DO CTEMP[J] := ' ';
1862        I:=I+1; CTEMP[I]:='$';
1863        IF (CS<>'    ') THEN ADD_TERM(CS,4);
1864        I:=I+1; CTEMP[I]:='$';
1865        IF (CML<>BLANK2) THEN ADD_LIST(CML,26);
1866        I:=I+1; CTEMP[I]:='$';
1867        IF (CA<>'    ') THEN ADD_TERM(CA,4);
1868        I:=I+1; CTEMP[I]:='$';
1869        IF (CRT <> BLANK2) THEN ADD_LIST(CRT,26);
1870        I:=I+1; CTEMP[I]:='$';
1871        IF (CT<>'    ') THEN ADD_TERM(CT,4);
1872        I:=I+1; CTEMP[I]:=';'
1873      END;
1874
1875
1876    "*************************************************************************
1877    *  UP_CODETAB:PROC_TAB:ARRAY[1..9999] OF CHAR                           *
1878    *              INSET ONE SMART STATEMENT TO PROC_TAB                    *
1879    *************************************************************************"
1880
1881    PROCEDURE UP_CODETAB(VAR PTR:INTEGER;VAR PTABLE:PROC_TABLE;
1882                         TEMP_C:CODE_TYPE);
1883      VAR I : INTEGER;
1884      BEGIN
1885        I := 1;
1886        REPEAT
1887          PTR := PTR + 1;
1888          PTABLE[PTR] := TEMP_C[I];
1889          I := I + 1
1890        UNTIL (TEMP_C[I] = ' ') OR (I = 71)
1891      END;
1892
1893
1894    "*************************************************************************
1895    *  CONTDCL:                                                             *
1896    *    1.HANDLE THE CONSTANT DECLARATION                                  *
1897    *    2.FILL UP THIS CONSTANT IN DICT_TAB'S LONGEVITY,ATTRIBUTE          *
1898    *      AND VALUEPTR                                                     *
1899    *************************************************************************"
1900
1901    PROCEDURE CONTDCL(VAR MKCT:INTEGER);
1902      VAR BECONTDCL,ITEXIST:BOOLEAN; APTR,K:INTEGER;
1903      BEGIN
1904        MK0:=0; MK1:=1;
1905        WHILE(AUNIT.CLAS='CONT   ') DO BEGIN
1906          BECONTDCL:=TRUE;
1907          GET_TOK(AUNIT,MK0,MK1,AHKY);
1908          WHILE (BECONTDCL=TRUE) DO BEGIN
1909            IF (AUNIT.CLAS='ID     ') THEN BEGIN
1910               K:=2;
1911               IF (K=MKCT)
1912                  THEN LEVEL_ID_TAB[CURRENT_LEVEL,K]:=AHKY
1913                  ELSE BEGIN ITEXIST:=FALSE;
```

```
1914                REPEAT
1915                  IF (LEVEL_ID_TAB[CURRENT_LEVEL,K]=AHKY)
1916                     THEN ITEXIST:=TRUE ELSE K:=K+1
1917                UNTIL (K=MKCT) OR (ITEXIST);
1918                IF (K=MKCT) THEN LEVEL_ID_TAB[CURRENT_LEVEL,K]:=AHKY
1919                END;
1920                IF (ITEXIST=FALSE) THEN MKCT:=MKCT+1;
1921                APTR:=AUNIT.INDEX;
1922                ID_TAB[AUNIT.INDEX].FLAG:=1;
1923                GET_TOK(AUNIT,MK0,MK1,AHKY)
1924                END ELSE BEGIN
1925                ERROR(CURNT_ST_NO,'CONTDCL SYNTAX ERROR');
1926                REPEAT
1927                  GET_TOK(AUNIT,MK0,MK1,AHKY)
1928                UNTIL (AUNIT.CLAS=';      ')OR(AUNIT.CLAS='ID     ');
1929                IF (AUNIT.CLAS=';      ')
1930                   THEN CURNT_ST_NO:=SUCC(CURNT_ST_NO);
1931                IF (AUNIT.CLAS='ID     ') THEN BEGIN
1932                   ID_TAB[AUNIT.INDEX].FLAG:=1;
1933                   APTR:=AUNIT.INDEX;
1934                   GET_TOK(AUNIT,MK0,MK1,AHKY)
1935                   END
1936             END;
1937          IF (AUNIT.CLAS='COMP_OP')&(AUNIT.INDEX=1)
1938             THEN GET_TOK(AUNIT,MK0,MK1,AHKY)
1939             ELSE ERROR(CURNT_ST_NO,'MISSING EQUAL SIGN  ');
1940          IF (AUNIT.CLAS='CONST  ') THEN BEGIN
1941             DICT_TAB[ID_TAB[APTR].DIDX].LONGEVITY:=1;
1942             DICT_TAB[ID_TAB[APTR].DIDX].ATTRIBUTE:=
1943             FIX_TAB[AUNIT.INDEX].TAG;
1944             DICT_TAB[ID_TAB[APTR].DIDX].VALUE_PTR:=AUNIT.INDEX;
1945             GET_TOK(AUNIT,MK0,MK1,AHKY)
1946             END ELSE BEGIN
1947             ERROR(CURNT_ST_NO,'CONTDCL SYNTAX ERROR');
1948             REPEAT
1949               GET_TOK(AUNIT,MK0,MK1,AHKY)
1950             UNTIL (AUNIT.CLAS=';      ')
1951             END;
1952          IF (AUNIT.CLAS=';      ')
1953             THEN CURNT_ST_NO:=SUCC(CURNT_ST_NO)
1954             ELSE BEGIN
1955             ERROR(CURNT_ST_NO,'MISSING SEMICOLON   ');
1956             REPEAT
1957               GET_TOK(AUNIT,MK0,MK1,AHKY)
1958             UNTIL (AUNIT.CLAS=';      ')
1959             END;
1960          GET_TOK(AUNIT,MK0,MK1,AHKY);
1961          IF (AUNIT.CLAS<>'ID     ') THEN BECONTDCL:=FALSE
1962        END"WHILE(BECONTDCL=TRUE)"
1963      END"WHILE(AUNIT.CLAS='CONT   ')"
1964    END;
1965
1966
1967  "*****************************************************************
1968  *  VARDCL:                                                      *
1969  *    1.HANDLE VARIABLES DECLARATION                             *
1970  *    2.FIX UP : ID_TAB'S FLAG,DICT_TAB'S ATTRIBUTE AND LONGEVITY *
1971  *****************************************************************"
```

```
1972
1973    PROCEDURE VARDCL(IJ:INTEGER;TTII,TTCC:THREE_ARY;AMKCT:INTEGER);
1974      VAR GETTING,BEVARDCL : BOOLEAN;
1975          LONGE:VALUETYPE; ATTRI:VALUETYPE;
1976          ID_STACK : ARRAY[1..10] OF INTEGER;
1977          ITOP,I,K,II : INTEGER;
1978          PP:PID_TAB_ENTRY; PD:DICT_TAB_ENTRY;
1979          STOPIT,ASTOP,SM_ERR,ITEXIST:BOOLEAN;
1980      BEGIN
1981        MK0:=0; MK1:=1; II:=AMKCT;
1982        STOPIT:=FALSE; ASTOP:=FALSE;
1983        SM_ERR:=FALSE; ITEXIST:=FALSE;
1984        WHILE (AUNIT.CLAS = 'VAR     ') DO BEGIN
1985          BEVARDCL := TRUE;
1986          GET_TOK(AUNIT,MK0,MK1,AHKY);
1987          WHILE (BEVARDCL) DO BEGIN
1988            GETTING:=TRUE; ITOP:=0;
1989            WHILE (GETTING) DO BEGIN
1990              IF (AUNIT.CLAS = 'ID      ') THEN BEGIN
1991                 K:=2;
1992                 IF (K=II)
1993                    THEN LEVEL_ID_TAB[CURRENT_LEVEL,K]:=AHKY
1994                    ELSE BEGIN ITEXIST:=FALSE;
1995                    REPEAT
1996                      IF (LEVEL_ID_TAB[CURRENT_LEVEL,K]=AHKY)
1997                         THEN ITEXIST:=TRUE ELSE K:=K+1
1998                    UNTIL (K=II) OR (ITEXIST);
1999                    IF (K=II) THEN LEVEL_ID_TAB[CURRENT_LEVEL,II]:=AHKY
2000                    END;
2001              IF (ITEXIST=FALSE) THEN II:=II+1;
2002              ID_TAB[AUNIT.INDEX].FLAG:=1;
2003              ITOP := ITOP + 1;
2004              ID_STACK[ITOP] := AUNIT.INDEX;
2005              GET_TOK(AUNIT,MK0,MK1,AHKY)
2006              END ELSE BEGIN
2007              ERROR(CURNT_ST_NO,'VARDCL SYNTAX ERROR ');
2008              REPEAT
2009                GET_TOK(AUNIT,MK0,MK1,AHKY)
2010              UNTIL (AUNIT.CLAS = 'ID      ') OR (AUNIT.CLAS =
2011                                ',       ') OR (AUNIT.CLAS = ':       ')
2012              END;
2013              IF (AUNIT.CLAS = ',       ')
2014                 THEN GET_TOK(AUNIT,MK0,MK1,AHKY)
2015                 ELSE IF (AUNIT.CLAS = ':       ')
2016                         THEN BEGIN
2017                         GETTING := FALSE;
2018                         GET_TOK(AUNIT,MK0,MK1,AHKY) END
2019            END "WHILE";
2020            IF (AUNIT.CLAS='REAL    ')OR(AUNIT.CLAS='INTEGER') OR
2021               (AUNIT.CLAS='CHAR    ')OR(AUNIT.CLAS='BOOLEAN')
2022               THEN BEGIN
2023               ATTRI:=AUNIT.INDEX;
2024               GET_TOK(AUNIT,MK0,MK1,AHKY);
2025               IF (AUNIT.CLAS=',       ')
2026                  THEN GET_TOK(AUNIT,MK0,MK1,AHKY)
2027               END ELSE ATTRI:=2;
2028            IF (AUNIT.CLAS='FIX     ')OR(AUNIT.CLAS='DYNAMIC') OR
2029               (AUNIT.CLAS='FLUID   ')OR(AUNIT.CLAS='STATIC ')
```

```
2030              THEN BEGIN
2031              LONGE:=AUNIT.INDEX; GET_TOK(AUNIT,MK0,MK1,AHKY)
2032              END ELSE LONGE:=3;
2033            REPEAT
2034              DICT_TAB[ID_TAB[ID_STACK[ITOP]].DIDX].ATTRIBUTE := ATTRI;
2035              DICT_TAB[ID_TAB[ID_STACK[ITOP]].DIDX].LONGEVITY := LONGE;
2036              I:=ID_TAB[ID_STACK[ITOP]].DIDX;
2037              IF (LONGE=2) THEN BEGIN
2038                 DICT_TAB[I].VALUE_PTR:=STA_COUNTER;
2039                 STA_COUNTER:=SUCC(STA_COUNTER) END;
2040              IF (LONGE=3) THEN BEGIN
2041                 DICT_TAB[I].VALUE_PTR:=DYN_COUNTER;
2042                 DYN_COUNTER:=SUCC(DYN_COUNTER) END;
2043              IF (LONGE=4) THEN BEGIN
2044                 DICT_TAB[I].VALUE_PTR:=FLD_COUNTER;
2045                 FLD_COUNTER:=SUCC(FLD_COUNTER) END;
2046              ITOP := ITOP - 1
2047            UNTIL (ITOP = 0);
2048            IF (AUNIT.CLAS = ';       ')
2049               THEN BEGIN
2050               GET_TOK(AUNIT,MK0,MK1,AHKY);
2051               IF (AUNIT.CLAS<>'ID      ') THEN BEVARDCL:=FALSE
2052               END ELSE ERROR(CURNT_ST_NO,'MISSING SEMICOLON    ');
2053            CURNT_ST_NO:=SUCC(CURNT_ST_NO)
2054          END "WHILE"
2055        END ;"WHILE "
2056        "FOR I:=2 TO 7 DO CHECK_NUM1(LEVEL_ID_TAB[CURRENT_LEVEL,I]);
2057        WRITE(NL);
2058        WRITESTMT('ID_TABLE:');
2059        FOR I:=1 TO 60 DO BEGIN
2060            CHECK_NUM1(ID_TAB[I].LEVEL_NO);
2061            CHECK_NUM1(ID_TAB[I].DIDX);
2062            CHECK_NUM1(ID_TAB[I].NEXT_PTR);
2063            CHECK_NUM1(ID_TAB[I].FLAG);
2064            WRITE(NL) END;"
2065        IF (PROGRAM_HEAD=FALSE) THEN BEGIN
2066           I:=0;
2067           PP:=PID_TAB[IJ];
2068           REPEAT
2069             I:=I+1;
2070             IF(PP.OUPUTL[I]=0) THEN
2071               STOPIT:=TRUE ELSE BEGIN
2072               PD:=DICT_TAB[PP.OUPUTL[I]];
2073               IF(PD.LONGEVITY=1) THEN BEGIN
2074                 SM_ERR:=TRUE;
2075                 ERROR(LEVEL_ID_TAB[CURRENT_LEVEL,1],'ILLEGAL OUPUT PARM  ')
2076                 END
2077               END
2078           UNTIL (STOPIT=TRUE) OR (SM_ERR=TRUE)
2079           END;
2080        I:=1;
2081     IF (TTII[I]<>0) THEN
2082        REPEAT
2083          IF (DICT_TAB[TTII[I]].ATTRIBUTE<>1)
2084             THEN ASTOP:=TRUE ELSE I:=I+1
2085        UNTIL (ASTOP=TRUE) OR (I=4) OR (TTII[I]=0);
2086        I:=1;
2087     IF (TTCC[I]<>0) THEN
```

```
2088        REPEAT
2089          IF (FIX_TAB[TTCC[I]].TAG<>1)
2090             THEN ASTOP:=TRUE ELSE I:=I+1
2091        UNTIL (ASTOP=TRUE) OR (I=4) OR (TTCC[I]=0);
2092        IF (ASTOP=TRUE) THEN
2093          ERROR(LEVEL_ID_TAB[CURRENT_LEVEL,1],'TERMI TYPE CONFLICT ')
2094      END;
2095
2096
2097   "*********************************************************************
2098   *  REQUEST:                                                         *
2099   *    REPEAT CALL                                                    *
2100   *      1.STIMU & RETURN 'CSNO':INDEX OF CURNT STIMU. IN STIMU_TAB   *
2101   *      2.FIX UP 'CANO':THE OPERATOR OR INDEX OF CURNT PID IN DICT_TAB*
2102   *      3.PARM_ARG & RETURN 'CRNO','CMNO':OUTPUTLIST,INPUTLIST       *
2103   *      4.TERMI & RETURN 'CTNO':INDEX OF CURNT TERMI. IN TERMI_TAB   *
2104   *      5.CREATE TO COLLECT THE ABOVE FIVE PARTS  AND RETURN ONE     *
2105   *        SMART INTERMEDIATE CODE STATEMENT                          *
2106   *      6.UP_CODETAB TO INSERT THIS STATEMENT TO PROC_TAB            *
2107   *      7.CURNT_ST_NO:=SUCC(CURNT_ST_NO)                             *
2108   *    UNTIL (AUNIT.CLAS='END    ')                                  *
2109   *********************************************************************"
2110
2111   PROCEDURE REQUESTS;
2112     VAR ERR_MARK:BOOLEAN; PIDPT:INTEGER; AAOP:TWOCH;
2113         TII,TCC:THREE_ARY; I,EX_ST_NO:INTEGER;
2114     BEGIN
2115       MK0:=1; MK1:=0; EX_ST_NO:=0;
2116       WHILE (AUNIT.CLAS = '[      ')OR(AUNIT.CLAS = 'PID    ') OR
2117             (AUNIT.CLAS = 'OP     ')OR(AUNIT.CLAS = ':=     ') DO BEGIN
2118         EX_ST_NO:=SUCC(EX_ST_NO);
2119         IF (AUNIT.CLAS = '[      ')
2120            THEN STIMU(ERR_MARK,CSNO) ELSE CSNO:='    ';
2121         IF (AUNIT.CLAS = 'PID    ') THEN BEGIN
2122            CONV_NUM(CANO,PID_TAB[AUNIT.INDEX].DIDX);
2123            PIDPT:=AUNIT.INDEX; AAOP:='  ';
2124            GET_TOK(AUNIT,MK0,MK1,AHKY)
2125            END ELSE
2126            IF (AUNIT.CLAS='OP     ')OR(AUNIT.CLAS=':=     ')
2127               THEN BEGIN
2128               CANO:='    '; PIDPT:=0;
2129               CANO[1]:=OP_TAB[AUNIT.INDEX,1];
2130               AAOP[1]:=OP_TAB[AUNIT.INDEX,1];
2131               CANO[2]:=OP_TAB[AUNIT.INDEX,2];
2132               AAOP[2]:=OP_TAB[AUNIT.INDEX,2];
2133               GET_TOK(AUNIT,MK0,MK1,AHKY)
2134               END ELSE BEGIN
2135               ERROR(CURNT_ST_NO,'MISSING ACTION NAME ');
2136               ERR_MARK:=TRUE; PIDPT:=0;
2137               CANO:='    '; AAOP:='  '
2138               END;
2139         IF (AUNIT.CLAS='(      ')
2140            THEN PARM_ARG(PIDPT,1,AAOP,ERR_MARK,CRNO,CMNO);
2141         IF (AUNIT.CLAS = '[      ') THEN TERMI(ERR_MARK,CTNO,1,TII,TCC)
2142            ELSE CTNO:='    ';
2143         IF (AUNIT.CLAS=';      ')
2144            THEN GET_TOK(AUNIT,1,0,AHKY)
2145            ELSE IF (AUNIT.CLAS<>'END    ') THEN BEGIN
```

```
2146                    ERROR(CURNT_ST_NO,'MISSING SEMICOLON   ');
2147                    FOR I:=1 TO 10 DO SYNC_TAB[I]:=BLANK;
2148                    SYNC_TAB[1]:='OP     ';
2149                    SYNC_TAB[2]:=':=     ';
2150                    SYNC_TAB[3]:=';      ';
2151                    SYNC_TAB[4]:='PID    ';
2152                    SYNC(SYNC_TAB,4);
2153                    IF (AUNIT.CLAS=';      ')
2154                       THEN GET_TOK(AUNIT,MK0,MK1,AHKY);
2155                    IF (AUNIT.CLAS='BEGIN  ')
2156                       THEN GET_TOK(AUNIT,1,0,AHKY)
2157                    END;
2158          CURNT_ST_NO:=SUCC(CURNT_ST_NO);
2159          CREATE_CODE(CTMP,CSNO,CANO,CTNO,CMNO,CRNO);
2160          UP_CODETAB(CODEIDX,PROC_TAB,CTMP)
2161        END "WHILE";
2162        DYNAM_TAB[DY_STACK[P_TOP-1]].TAG:=1;
2163        DYNAM_TAB[DY_STACK[P_TOP-1]].IVALUE:=EX_ST_NO
2164      END;
2165
2166
2167    "*************************************************************************
2168    *  ACTIONDCL:                                                           *
2169    *    1.DEFINE THE SAME LEVEL'S PROCEDURES                               *
2170    *    2.MARK PROCEDURE'S START_PT & INSERT '!' TO PROC_TAB AT THE END    *
2171    *      OF THE PROCEDURE AND FIX UP PID_TAB'S LENG                       *
2172    *************************************************************************"
2173
2174    PROCEDURE SAME_PROC(AAPID:INTEGER;ATI,ATC:THREE_ARY); FORWARD;
2175    PROCEDURE ACTIONDCL;
2176      VAR I,APID,START_PT : INTEGER;
2177          B_V : BOOLEAN;
2178          TI,TC : THREE_ARY;
2179          CCXX : CINDEX;
2180          CTMNO : ARRAY[1..25] OF CHAR;
2181          II,J : INTEGER;
2182
2183      BEGIN
2184        WHILE (AUNIT.CLAS = 'PROC   ') DO BEGIN
2185          MK0:=1; MK1:=1;
2186          GET_TOK(AUNIT,MK0,MK1,AHKY);
2187          IF (AUNIT.CLAS = '[      ')
2188             THEN STIMU(B_V,CSNO) ELSE CSNO:='    ';
2189          IF (AUNIT.CLAS = 'PID    ') THEN BEGIN
2190             CONV_NUM(CANO,PID_TAB[AUNIT.INDEX].DIDX);
2191             APID:=AUNIT.INDEX;
2192             P_STACK[P_TOP] := AUNIT.INDEX;
2193             GET_TOK(AUNIT,MK0,MK1,AHKY) END ELSE BEGIN
2194             P_STACK[P_TOP] := 0;
2195             ERROR(CURNT_ST_NO,'MISSING ACTION NAME ');
2196             FOR I := 1 TO 10 DO SYNC_TAB[I] := BLANK;
2197             SYNC_TAB[1] := '(      ';
2198             SYNC_TAB[2] := '[      ';
2199             SYNC_TAB[3] := ';      ';
2200             SYNC(SYNC_TAB,3) END;
2201          DICT_TAB[CURRENT_DIDX].USER:='%NO_ST ';
2202          DICT_TAB[CURRENT_DIDX].CONTEXT:=CURRENT_CONTEXT;
2203          DICT_TAB[CURRENT_DIDX].ATTRIBUTE:=1;
```

```
2204          DICT_TAB[CURRENT_DIDX].LONGEVITY:=3;
2205          DICT_TAB[CURRENT_DIDX].VALUE_PTR:=DYN_COUNTER;
2206          DY_STACK[P_TOP]:=DYN_COUNTER;
2207          DYN_COUNTER:=SUCC(DYN_COUNTER);
2208          CONV_NUM(CCXX,CURRENT_DIDX);
2209          CURRENT_DIDX:=SUCC(CURRENT_DIDX);
2210          CURRENT_LEVEL:=SUCC(CURRENT_LEVEL);
2211        P_TOP := SUCC(P_TOP);
2212        LEVEL_ID_TAB[CURRENT_LEVEL,0]:=CURNT_COLL;
2213        LEVEL_ID_TAB[CURRENT_LEVEL,1]:=CURNT_ST_NO;
2214        IF (AUNIT.CLAS='(       ')
2215           THEN PARM_ARG(APID,0,'  ',B_V,CRNO,CMNO);
2216        CTMNO:='%D                          ';
2217        I:=1;
2218        REPEAT
2219          CTMNO[I+2]:=CCXX[I];
2220        I:=I+1
2221        UNTIL (I=5) OR (CCXX[I]=' ');
2222        CTMNO[I+2]:=',';
2223        II:=I+2;
2224        FOR J:=1 TO (25-II) DO CTMNO[J+II]:=CMNO[J];
2225        FOR J:=1 TO 25 DO CMNO[J]:=CTMNO[J];
2226        IF (AUNIT.CLAS = '[       ') THEN TERMI(B_V,CTNO,0,TI,TC)
2227           ELSE CTNO:='    ';
2228        START_PT:=CODEIDX+1;
2229        SAME_PROC(APID,TI,TC);
2230        IF (AUNIT.CLAS = ';       ')
2231           THEN GET_TOK(AUNIT,MK0,MK1,AHKY)
2232           ELSE BEGIN
2233                  ERROR(CURNT_ST_NO,'MISSING ; AFTER END ');
2234                  FOR I := 1 TO 10 DO SYNC_TAB[I] := BLANK;
2235                  SYNC_TAB[1] := 'PROC   ';
2236                  SYNC_TAB[2] := '[      ';
2237                  SYNC_TAB[3] := 'PID    ';
2238                  SYNC_TAB[4] := 'OP     ';
2239                  SYNC_TAB[5] := 'BEGIN  ';
2240                  SYNC(SYNC_TAB,5)
2241                END;
2242         CURNT_ST_NO:=SUCC(CURNT_ST_NO);
2243         CODEIDX:=SUCC(CODEIDX);
2244         PROC_TAB[CODEIDX]:='!';
2245         DICT_TAB[PID_TAB[APID].DIDX].VALUE_PTR:=START_PT;
2246         PID_TAB[APID].LENG:=CODEIDX-START_PT+1
2247       END "WHILE"
2248     END;
2249
2250
2251   "*************** MAIN_HEAD:HANDLE 'PROGRAM MAIN' ***********************"
2252   "* MAIN_HEAD:HANDLE 'PROGRAM PID' & MARK '.' AT THE BEGIN OF PROGRAM *"
2253   PROCEDURE MAIN_HEAD;
2254     VAR I,J : INTEGER;
2255         CCXX:CINDEX;
2256     BEGIN
2257       MK0:=1;  MK1:=1; READ(CH);
2258       LEVEL_ID_TAB[1,0]:=51; LEVEL_ID_TAB[1,1]:=1;
2259       DICT_TAB[1].VALUE_PTR:=1;
2260       GET_TOK(AUNIT,MK0,MK1,AHKY);
2261       IF AUNIT.CLAS = 'PROGRAM'
```

```
2262          THEN GET_TOK(AUNIT,1,MK1,AHKY)
2263          ELSE BEGIN
2264                 ERROR(CURNT_ST_NO,'MISSING KEY PROGRAM ');
2265                 FOR I := 1 TO 10 DO SYNC_TAB[I] := BLANK;
2266                 SYNC_TAB[1] := 'PID    ';
2267                 SYNC_TAB[2] := ';      ';
2268                 SYNC(SYNC_TAB,2)
2269               END;
2270       IF AUNIT.CLAS = 'PID    '
2271          THEN BEGIN
2272                 P_STACK[P_TOP] := AUNIT.INDEX;
2273                 GET_TOK(AUNIT,MK0,MK1,AHKY)
2274               END
2275          ELSE BEGIN
2276                 P_STACK[P_TOP] := 0;
2277                 ERROR(CURNT_ST_NO,'MISSING PROGRAM NAME');
2278                 FOR I := 1 TO 10 DO SYNC_TAB[I] := BLANK;
2279                 SYNC_TAB[1] := ';      ';
2280                 SYNC_TAB[2] := 'VAR    ';
2281                 SYNC_TAB[3] := 'PROC   ';
2282                 SYNC_TAB[4] := '[      ';
2283                 SYNC_TAB[5] := 'PID    ';
2284                 SYNC_TAB[7] := ':=     ';
2285                 SYNC(SYNC_TAB,7)
2286               END;
2287       CSNO:='    '; CTNO:='    '; CANO:='1   ';
2288       CRNO:=BLANK2;
2289       DICT_TAB[CURRENT_DIDX].USER:='%NO_ST ';
2290       DICT_TAB[CURRENT_DIDX].CONTEXT:=CURRENT_CONTEXT;
2291       DICT_TAB[CURRENT_DIDX].ATTRIBUTE:=1;
2292       DICT_TAB[CURRENT_DIDX].LONGEVITY:=3;
2293       DICT_TAB[CURRENT_DIDX].VALUE_PTR:=DYN_COUNTER;
2294       DY_STACK[P_TOP]:=DYN_COUNTER;
2295       DYN_COUNTER:=SUCC(DYN_COUNTER);
2296       CONV_NUM(CCXX,CURRENT_DIDX);
2297       CURRENT_DIDX:=SUCC(CURRENT_DIDX);
2298       CMNO:='%D                        ';
2299       I:=0;
2300       REPEAT
2301         I:=I+1; CMNO[I+2]:=CCXX[I]
2302       UNTIL (I=4) OR (CCXX[I]=' ');
2303       CURRENT_LEVEL := SUCC(CURRENT_LEVEL);
2304       P_TOP := SUCC(P_TOP);
2305     END;
2306
2307
2308   "************************ ARRANGE_ID_TAB ****************************"
2309   PROCEDURE ARRANGE_ID_TAB;
2310     VAR I,K,IK : INTEGER;
2311     BEGIN
2312       I:=2;
2313       REPEAT
2314         K:=LEVEL_ID_TAB[CURRENT_LEVEL,I];
2315         REPEAT
2316           IF (ID_TAB[K].LEVEL_NO=CURRENT_LEVEL) THEN
2317              IF (ID_TAB[K].NEXT_PTR=0) THEN BEGIN
2318                 ID_TAB[K].DIDX:=0;
2319                 ID_TAB[K].LEVEL_NO:=0;
```

```
2320                ID_TAB[K].FLAG:=0
2321                END ELSE BEGIN
2322                IK:=ID_TAB[K].NEXT_PTR;
2323                ID_TAB[K].LEVEL_NO:=0;
2324                ID_TAB[K].DIDX:=0;
2325                ID_TAB[K].NEXT_PTR:=0;
2326                ID_TAB[K].FLAG:=0; K:=IK END
2327              ELSE K:=ID_TAB[K].NEXT_PTR
2328          UNTIL(K=0);
2329          I:=I+1
2330        UNTIL(LEVEL_ID_TAB[CURRENT_LEVEL,I]=0)
2331      END;
2332
2333
2334   "*************************************************************************
2335   *  SAME_PROC:                                                           *
2336   *    1.INSERT ONE SMART STATEMENT TO PROC_TAB                           *
2337   *    2.CURNT_ST_NO:=SUCC(CURNT_ST_NO)                                   *
2338   *    3.CALL CONTDCL,VARDCL,ACTIONDCL,REQUESTS                           *
2339   *    4.CURRENT_LEVEL:=CURRRENT_LEVEL-1                                  *
2340   *************************************************************************"
2341
2342   PROCEDURE SAME_PROC"(AAPID:INTEGER;ATI,ATC:THREE_ARY)";
2343     VAR I,MKCTER : INTEGER;
2344     BEGIN
2345       MK0:=1; MK1:=0;
2346       IF (AUNIT.CLAS = ';       ')
2347          THEN GET_TOK(AUNIT,MK0,MK1,AHKY)
2348          ELSE ERROR(CURNT_ST_NO,'MISSING SEMICOLON    ');
2349       CURNT_ST_NO := SUCC(CURNT_ST_NO);
2350       IF (PROGRAM_HEAD=TRUE) THEN BEGIN
2351          CODEIDX:=SUCC(CODEIDX);
2352          PROC_TAB[CODEIDX]:='.';
2353          PROGRAM_HEAD:=FALSE
2354          END;
2355       CREATE_CODE(CTMP,CSNO,CANO,CTNO,CMNO,CRNO);
2356       UP_CODETAB(CODEIDX,PROC_TAB,CTMP);
2357       FOR I:=2 TO 20 DO LEVEL_ID_TAB[CURRENT_LEVEL,I]:=0;
2358       MKCTER := 2;
2359       IF (AUNIT.CLAS = 'CONT    ') THEN CONTDCL(MKCTER);
2360       IF (AUNIT.CLAS = 'VAR     ') THEN VARDCL(AAPID,ATI,ATC,MKCTER);
2361       IF (AUNIT.CLAS = 'PROC    ') THEN ACTIONDCL;
2362       IF (AUNIT.CLAS = 'BEGIN   ')
2363           THEN GET_TOK(AUNIT,1,0,AHKY) ELSE BEGIN
2364           ERROR(CURNT_ST_NO,'MISSING KEY BEGIN    ');
2365           FOR I:=1 TO 10 DO SYNC_TAB[I]:=BLANK;
2366           SYNC_TAB[1]:='VAR     ';
2367           SYNC_TAB[2]:=';       ';
2368           SYNC_TAB[3]:='BEGIN   ';
2369           SYNC_TAB[4]:='OP      ';
2370           SYNC_TAB[5]:=':=      ';
2371           SYNC_TAB[6]:='[       ';
2372           SYNC(SYNC_TAB,6);
2373           IF (AUNIT.CLAS=';       ') THEN BEGIN
2374              CURNT_ST_NO:=SUCC(CURNT_ST_NO);
2375              GET_TOK(AUNIT,MK0,MK1,AHKY)
2376              END;
2377           IF (AUNIT.CLAS='BEGIN   ')
```

```
2378              THEN GET_TOK(AUNIT,1,0,AHKY)
2379             END;
2380        IF (AUNIT.CLAS = '[       ') OR (AUNIT.CLAS = 'PID     ') OR
2381           (AUNIT.CLAS = 'OP      ') OR (AUNIT.CLAS = ':=      ')
2382           THEN REQUESTS;
2383        ARRANGE_ID_TAB;
2384        CURRENT_LEVEL := CURRENT_LEVEL - 1;
2385        P_TOP := P_TOP - 1;
2386        IF AUNIT.CLAS = 'END     '
2387           THEN GET_TOK(AUNIT,1,0,AHKY)
2388           ELSE BEGIN
2389                  ERROR(CURNT_ST_NO,'MISSING KEY END      ');
2390                  FOR I := 1 TO 10 DO SYNC_TAB[I] := BLANK;
2391                  SYNC_TAB[1] := 'PID     ';
2392                  SYNC_TAB[2] := ';       ';
2393                  SYNC_TAB[3] := '.       ';
2394                  SYNC(SYNC_TAB,3)
2395                END;
2396        IF (AUNIT.CLAS = 'PID     ') THEN BEGIN
2397           IF (P_STACK[P_TOP] <> AUNIT.INDEX)
2398              THEN ERROR(CURNT_ST_NO,'PROC-NAME NOT MATCH ');
2399           GET_TOK(AUNIT,MK0,MK1,AHKY)
2400           END
2401      END;
2402
2403
2404    "************************ WRITE_TABS ***********************************"
2405    PROCEDURE WRITE_TABS;
2406      VAR I,K : INTEGER;
2407      BEGIN
2408        WRITE(NL); WRITE(NL);
2409        "WRITESTMT('ID_TABLE:'); WRITE(NL);
2410        FOR I:=0 TO 100 DO BEGIN
2411            CHECK_NUM1(ID_TAB[I].LEVEL_NO);
2412            CHECK_NUM1(ID_TAB[I].DIDX);
2413            CHECK_NUM1(ID_TAB[I].NEXT_PTR);
2414            CHECK_NUM1(ID_TAB[I].FLAG);
2415            WRITE(NL) END;"
2416       "WRITESTMT('PID_TAB: '); WRITE(NL);
2417       FOR I:=0 TO CURRENT_PIDX DO BEGIN
2418           CHECK_NUM1(PID_TAB[I].DIDX);
2419           CHECK_NUM1(PID_TAB[I].OLEN);
2420           CHECK_NUM1(PID_TAB[I].ILEN);
2421           CHECK_NUM1(PID_TAB[I].LENG);
2422           FOR K:=1 TO 5 DO BEGIN
2423               WRITE(PID_TAB[I].INPUTL[K].FLAG);
2424               CHECK_NUM1(PID_TAB[I].INPUTL[K].INDX)
2425               END;
2426           FOR K:=1 TO 5 DO CHECK_NUM1(PID_TAB[I].OUPUTL[K]);
2427           WRITE(NL) END;"
2428       WRITESTMT('DICT_TAB:'); WRITE(NL);
2429       FOR I:=0 TO (CURRENT_DIDX-1) DO BEGIN
2430           FOR K:=0 TO 6 DO WRITE(DICT_TAB[I].USER[K]);
2431           FOR K:=0 TO 9 DO CHECK_NUM1(DICT_TAB[I].CONTEXT[K]);
2432           CHECK_NUM1(DICT_TAB[I].NEXT_SEQ_NO);
2433           CHECK_NUM1(DICT_TAB[I].ATTRIBUTE);
2434           CHECK_NUM1(DICT_TAB[I].LONGEVITY);
2435           CHECK_NUM1(DICT_TAB[I].COPIES_NO);
```

```
2436          CHECK_NUM1(DICT_TAB[I].VALUE_PTR);
2437          WRITE(NL) END;
2438       WRITE(NL); WRITE(NL);
2439       WRITESTMT('FIX_TAB: '); WRITE(NL);
2440       FOR I:=0 TO FIX_COUNTER DO BEGIN
2441          CHECK_NUM1(FIX_TAB[I].TAG);
2442          CHECK_NUM1(FIX_TAB[I].IVALUE);
2443          WRITE(NL) END;
2444       WRITE(NL); WRITE(NL);
2445       WRITESTMT('DYNA_TAB:'); WRITE(NL);
2446       FOR I:=0 TO DYN_COUNTER DO BEGIN
2447          CHECK_NUM1(DYNAM_TAB[I].SEQ_NO);
2448          CHECK_NUM1(DYNAM_TAB[I].NEXT_SEQ_NO);
2449          CHECK_NUM1(DYNAM_TAB[I].TAG);
2450          CHECK_NUM1(DYNAM_TAB[I].IVALUE);
2451          WRITE(NL) END;
2452       WRITE(NL); WRITE(NL);
2453       WRITESTMT('STIM_TAB:'); WRITE(NL);
2454       FOR I:=1 TO STIMU_IDX DO WRITE(STIMU_TAB[I]); WRITE(NL);
2455       WRITE(NL);
2456       WRITESTMT('TERM_TAB:'); WRITE(NL);
2457       FOR I:=1 TO TERMI_IDX DO BEGIN
2458          FOR K:=1 TO 38 DO WRITE(TERMI_TAB[I,K]);
2459          WRITE(NL) END
2460       "WRITESTMT('LEVL_TAB'); WRRITE(NL);
2461       FOR I:=0 TO 5 DO BEGIN
2462          FOR K:=0 TO 20 DO CHECK_NUM1(LEVEL_ID_TAB[I,K]);
2463          WRITE(NL) END"
2464    END;
2465
2466
2467 "*********** MAIN_TAIL:END OF THE MAIN PROGRAM MARK '.' ***************"
2468 PROCEDURE MAIN_TAIL;
2469    VAR I,K : INTEGER;
2470    BEGIN
2471      MK0:=0; MK1:=0;
2472      IF (AUNIT.CLAS = '.      ') THEN BEGIN
2473          CODEIDX := SUCC(CODEIDX);
2474          PROC_TAB[CODEIDX] := '.';
2475          PID_TAB[1].LENG := CODEIDX;
2476          WRITE_TABS;
2477          K:=1;
2478          WRITE(NL); WRITE(NL);
2479          WRITESTMT('PROC_TAB:');
2480          REPEAT
2481            REPEAT WRITE(PROC_TAB[K]); K:=K+1 UNTIL(PROC_TAB[K]=' ');
2482            WRITE(NL)
2483          UNTIL(K>200);
2484          GET_TOK(AUNIT,MK0,MK1,AHKY) END ELSE
2485          ERROR(CURNT_ST_NO,'MISSING PERIOD      ');
2486      IF (AUNIT.CLAS[0] <> EM) THEN BEGIN
2487          ERROR(CURNT_ST_NO,'DELETE EXTRA CONTEXT');
2488          REPEAT WRITE(CH); READ(CH) UNTIL (CH=EM);
2489          WRITE(EM) END
2490    END;
2491
2492    "*BEGIN*"
2493    BEGIN
```

```
2494          INITIALIZE;
2495          MAIN_HEAD;
2496          SAME_PROC(1,EMPTY0,EMPTY0);
2497          MAIN_TAIL
2498        END.
2499
```

APPENDIX F SAMPLE RUNS

SAMPLE 1

SOURCE CODE

```
1       PROGRAM FIB;
2       CONT ANO=3;
3            BNO=9;
4       VAR RESULT,LASTRES:INTEGER,DYNAMIC;
5           N:INTEGER,STATIC;
6       PROC LOOP(RSLT.+1,LTRS.+1:RSLT.+0,LTRS.+0,N)[INC I(3,+1,"BNO"9)];
7          VAR I,RSLT,LTRS:INTEGER,DYNAMIC;
8              N:INTEGER,STATIC;
9          BEGIN
10           +(RSLT.+1:RSLT.+0,LTRS.+0);
11           :=(LTRS.+1:RSLT.+0)
12         END LOOP;
13      PROC REST(RESULT:N);
14         VAR RESULT,LASTRES:INTEGER,DYNAMIC;
15             N:INTEGER,STATIC;
16         BEGIN
17           :=(RESULT,LASTRES:1);
18           LOOP(RESULT.+1,LASTRES.+1:RESULT.+0,LASTRES.+0,N)
19         END REST;
20      BEGIN
21        :=(N:10);
22        [N<ANO] -(RESULT:N,1);
23        [N>=ANO] REST(RESULT:N)
24      END FIB.
```

```
PROGRAM FIB;
  CONT ANO=3;
       BNO=9;
  VAR RESULT,LASTRES:INTEGER,DYNAMIC;
      N:INTEGER,STATIC;
  PROC LOOP(RSLT.+1,LTRS.+1:RSLT.+0,LTRS.+0,N)[INC I(3,+1,"BNO"9)];
     VAR I,RSLT,LTRS:INTEGER,DYNAMIC;
         N:INTEGER,STATIC;
     BEGIN
       +(RSLT.+1:RSLT.+0,LTRS.+0);
       :=(LTRS.+1:RSLT.+0)
     END LOOP;
  PROC REST(RESULT:N);
     VAR RESULT,LASTRES:INTEGER,DYNAMIC;
         N:INTEGER,STATIC;
     BEGIN
       :=(RESULT,LASTRES:1);
       LOOP(RESULT.+1,LASTRES.+1:RESULT.+0,LASTRES.+0,N)
     END REST;
  BEGIN
    :=(N:10);
    [N<ANO] -(RESULT:N,1);
    [N>=ANO] REST(RESULT:N)
  END FIB.
```

DICT_TAB:

	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FIB	0	0	0	0	0	0	0	0	0	0	0	0	5	0	1
%NO_ST	1	0	0	0	0	0	0	0	0	0	0	1	3	0	1
ANO	1	0	0	0	0	0	0	0	0	0	0	1	1	0	1
BNO	1	0	0	0	0	0	0	0	0	0	0	1	1	0	2
RESULT	1	0	0	0	0	0	0	0	0	0	0	1	3	0	3
LASTRES	1	0	0	0	0	0	0	0	0	0	0	1	3	0	2
N	1	0	0	0	0	0	0	0	0	0	0	1	2	0	1
LOOP	1	0	0	0	0	0	0	0	0	0	0	0	5	0	13
%NO_ST	1	8	0	0	0	0	0	0	0	0	0	1	3	0	4
RSLT	1	8	0	0	0	0	0	0	0	0	0	1	3	0	6
LTRS	1	8	0	0	0	0	0	0	0	0	0	1	3	0	5
N	1	8	0	0	0	0	0	0	0	0	0	1	2	0	2
I	1	8	0	0	0	0	0	0	0	0	0	1	3	0	7
REST	1	8	0	0	0	0	0	0	0	0	0	0	5	0	101
%NO_ST	1	14	0	0	0	0	0	0	0	0	0	1	3	0	8
RESULT	1	14	0	0	0	0	0	0	0	0	0	1	3	0	10
N	1	14	0	0	0	0	0	0	0	0	0	1	2	0	3
LASTRES	1	14	0	0	0	0	0	0	0	0	0	1	3	0	9

FIX_TAB:

1	0
1	3
1	9
1	1
1	10
1	0

DYNA_TAB:

```
0    0    1    0
0    0    1    3
0    0    1    0
0    0    1    0
0    0    1    2
0    0    1    0
0    0    1    0
0    0    1    0
0    0    1    2
0    0    1    0
0    0    1    0
0    0    1    0
```

```
STIM_TAB:
D(7)<D(3);D(7)>=D(3);
```

```
TERM_TAB:
FOR D(13 ):=F(1  )TO F(2  )BY+F(3  )DO
```

```
PROC_TAB: .
 $$%D2$1$$;
 $$%D9,D10.+0,D11.+0,D12$8$10.+1,11.+1$1;
 $$D10.+0,D11.+0$+$10.+1$;
 $$D10.+0$:=$11.+1$;!
 $$%D15,D17$14$16$;
 $$F3$:=$16,18$;
 $$D16.+0,D18.+0,D17$8$16.+1,18.+1$;!
 $$F4$:=$7$;
 $1$D7,F3$-$5$;
 $11$D7$14$5$;.
```

SAMPLE 2

SOURCE CODE

```
1     FIB;
2      CONT ANO=3;
3           BNO=9;
4      VAR RESULT,LASTRES:INTEGER,DYNAMIC
5          N,RESULT:INTEGER,STATIC;
6      PROC LOOP(RSLT.+1,LTRS.+1:RSLT.+0,LTRS.+0,N)[INC I(3,+1,"BNO"9);
7         VAR I,LTRS:INTEGER,DYNAMIC;
8             N:INTEGER,STATIC;
9           +(RSLT.+1:RSLT.+0,LTRS.+0;
10          :=(LTRS.+1:RSLT.+0)
11        END LOOOP;
12     PROC REST(RESULT:N);
13         RESULT,LASTRES:INTEGER,DYNAMIC;
14            N:INTEGER,STATIC;
15        BEGIN
16          :=(RESULT,LASTRES:1);
17          LOOP(RESULT.+1,LASTRES.+1:RESULT.+0,LASTRES.+0,N)
18        END REST
19     BEGIN
20       :=(N:10);
21       [N<ANO] -(RESULT:N,1);
22       [N>=ANO] REST(RESULT:N,I)
23     END FIB. THIS IS A TEST PROGRAM
```

```
 FIB
STATEMENT 1     MISSING KEY PROGRAM
;
  CONT ANO=3;
       BNO=9;
  VAR RESULT,LASTRES:INTEGER,DYNAMIC
      N
STATEMENT 4     MISSING SEMICOLON
,RESULT
STATEMENT 5     REDEFINED IDENTIFIER
:INTEGER,STATIC;
  PROC LOOP(RSLT.+1,LTRS.+1:RSLT.+0,LTRS.+0,N)[INC I(3,+1,"BNO"9);
STATEMENT 6     TERMI SYNTAX ERROR

     VAR I,LTRS:INTEGER,DYNAMIC;
         N:INTEGER,STATIC;
       +
STATEMENT 9     MISSING KEY BEGIN
(RSLT
STATEMENT 9     UNDEFINED IDENTIFIER
.+1:RSLT
STATEMENT 9     UNDEFINED IDENTIFIER
.+0,LTRS.+0;
STATEMENT 9     PARM_ARGS SYNTAX_ERR

STATEMENT 9     MISSING RPARENTH

STATEMENT 9     WARNING:CONV INTEGER

       :=
STATEMENT 9     MISSING SEMICOLON
(LTRS.+1:RSLT
STATEMENT 10    UNDEFINED IDENTIFIER
.+0)
STATEMENT 10    OPRND TYPE CONFLICT

STATEMENT 10    OPRND TYPE CONFLICT

     END LOOOP
STATEMENT 11    UNDEFINED PROC_NAME

STATEMENT 11    PROC-NAME NOT MATCH
;
  PROC REST(RESULT:N);
       RESULT
STATEMENT 12    UNDEFINED PROC_NAME

STATEMENT 13    MISSING KEY BEGIN
,LASTRES:INTEGER,DYNAMIC;
         N
STATEMENT 14    UNDEFINED PROC_NAME
:
STATEMENT 14    MISSING SEMICOLON
INTEGER,STATIC;
     BEGIN
       :=(RESULT
STATEMENT 15    UNDEFINED IDENTIFIER
,LASTRES:1)
STATEMENT 15    OPRND TYPE CONFLICT
;
```

```
      LOOP(RESULT
STATEMENT 16   UNDEFINED IDENTIFIER
.+1,LASTRES.+1:RESULT
STATEMENT 16   UNDEFINED IDENTIFIER
.+0,LASTRES.+0,N
STATEMENT 16   UNDEFINED IDENTIFIER
)
     END REST
  BEGIN
STATEMENT 17   MISSING ; AFTER END

    :=(N
STATEMENT 18   UNDEFINED IDENTIFIER
:10)
STATEMENT 18   OPRND TYPE CONFLICT

STATEMENT 18   OPRND TYPE CONFLICT
;
    [N
STATEMENT 19   UNDEFINED IDENTIFIER
<ANO]
STATEMENT 19   OPRND TYPE CONFLICT
 -(RESULT
STATEMENT 19   UNDEFINED IDENTIFIER
:N
STATEMENT 19   UNDEFINED IDENTIFIER
,1)
STATEMENT 19   WARNING:CONV INTEGER
;
    [N
STATEMENT 20   UNDEFINED IDENTIFIER
>=ANO]
STATEMENT 20   OPRND TYPE CONFLICT
 REST(RESULT
STATEMENT 20   UNDEFINED IDENTIFIER
:N
STATEMENT 20   UNDEFINED IDENTIFIER
,I
STATEMENT 20   UNDEFINED IDENTIFIER
)
STATEMENT 20   INARGS_NO<>PARMS_NO

  END FIB.


DICT_TAB:
        0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
FIB     0   0   0   0   0   0   0   0   0   0   0   0   5   0   1
%NO_ST  1   0   0   0   0   0   0   0   0   0   0   1   3   0   1
ANO     1   0   0   0   0   0   0   0   0   0   0   1   1   0   1
BNO     1   0   0   0   0   0   0   0   0   0   0   1   1   0   2
RESULT  1   0   0   0   0   0   0   0   0   0   0   1   2   0   1
LASTRES1    0   0   0   0   0   0   0   0   0   0   1   3   0.  2
N       1   0   0   0   0   0   0   0   0   0   0   1   2   0   2
LOOP    1   0   0   0   0   0   0   0   0   0   0   0   5   0   13
%NO_ST  1   8   0   0   0   0   0   0   0   0   0   1   3   0   4
RSLT    1   8   0   0   0   0   0   0   0   0   0   0   0   0   0
LTRS    1   8   0   0   0   0   0   0   0   0   0   1   3   0   5
N       1   8   0   0   0   0   0   0   0   0   0   1   2   0   3
I       1   8   0   0   0   0   0   0   0   0   0   1   3   0   6
```

```
REST    1    8    0    0    0    0    0    0    0    0    0    0    5    0    101
%NO_ST  1    14   0    0    0    0    0    0    0    0    0    1    3    0    7
RESULT  1    14   0    0    0    0    0    0    0    0    0    0    0    0    0
N       1    14   0    0    0    0    0    0    0    0    0    0    0    0    0


FIX_TAB:
1    0
1    3
1    9
1    1
1    10
1    0


DYNA_TAB:
0    0    1    0
0    0    1    3
0    0    1    0
0    0    1    0
0    0    1    2
0    0    1    0
0    0    1    0
0    0    1    3
0    0    1    0


STIM_TAB:
D(17)<D(3);D(17)>=D(3);


TERM_TAB:
FOR D(13 ):=F(1   )TO F(2   )BY+F(3   )DO


PROC_TAB: .
 $$%D2$1$$;
 $$%D9,D10.+0,D11.+0,D12$8$10.+1,11.+1$1;
 $$D10.+0,D11.+0$+$10.+1$;
 $$D10.+0$:=$11.+1$;!
 $$%D15,D17$14$16$;
 $$%D15,D17$0$16$;
 $$F3$:=$16,6$;
 $$D16.+0,D6.+0,D17$8$16.+1,6.+1$;!
 $$F4$:=$17$;
 THIS
STATEMENT 21   UNDEFINED IDENTIFIER

STATEMENT 21   DELETE EXTRA CONTEXT
 IS A TEST PROGRAM
```

APPENDIX G
THE LISTING OF ERROR MESSAGES

SYNTACTIC ERROR MESSAGES:

1. contdcl syntax error
2. delete extra context
3. factor syntax error
4. missing action name
5. missing colon
6. missing equal sign
7. missing key begin
8. missing key end
9. missing key program
10. missing lparenth
11. missing period
12. missing program name
13. missing semicolon
14. missing rparenth
15. missing ; after end
16. name too long
17. no , sepat parm_args
18. number_error
19. parenthes not match
20. stimu syntax error
21. string_error
22. termi missing comma
23. termi missing rword
24. termi missing "("
25. termi missing ")"
26. termi syntax error

27. vardcl syntax error
28. v_exist syntax error

SEMANTICS ERROR MESSAGES:

1. expr factor error
2. illegal in_ouput_no
3. illegal input parm
4. illegal ouput parm
5. inargs_no<>parms_no
6. inargs_tp<>parms_tp
7. oprnd type conflict
8. outargs_no<>parms_no
9. outargs_tp<>parms_tp
10. proc_name not match
11. redefined identifier
12. redefined proc_name
13. termi type conflict
14. undefined identifier
15. undefined proc_name
16. warning:conv integer
17. wrong input parm_arg
18. wrong ouput parm_arg
19. wrong parm_arg

AN INHERENTLY CONCURRENT LANGUAGE : A TRANSLATOR

by

CHUNMEI LIOU SIANG

B.S. Chung-Yuan College of Science and Engineering, 1972

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1980

ABSTRACT

In most existing programming languages, programs are executed sequentially, and the written order of the code determines the order of execution of program statements. This complied with the fact only one processor was available at any one time in the past. Thus Yesterday's machine capabilities have limited us to sequential processing in our programming languages. A model for a class of languages capable of utilizing the hardware concurrency in a natural fashion has been developed. The sequencing in this model is determined by the "data-drive" principle with further programmer control exercised through conditions.

The first step of this research project precisely defines the project language in Backus-Naur Form which is mapped from the ACM model. Next, a translator was written to transform the source language into a form close to the concurrent model and acceptable as the input to a simulator which be written by another graduate student R. McBride. The ultimate purpose of this research is to study the underlying behavior of the concurrent model, ACM, through simulation of programs which written in the project language. The results of these simulations can be used to determine where improvements need to be made.