UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

THE IMPLEMENTATION OF A FRONT END PROCESSOR FOR A SUBSET OF ADA (1)

BY

JACQUELINE EPSTEIN

A THESIS

PREPARED UNDER THE SUPERVISION OF

PROF. K. J. MACGREGOR

IN FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE

(1) ADA IS A REGISTERED TRADEMARK OF THE U.S. GOVERNMENT, ADA JOINT PROGRAM OFFICE

THIS THESIS IS SPONSORED IN PART BY THE

COUNCIL FOR SCIENTIFIC AND INDUSTRIAL RESEARCH

APRIL, 1983

ACKNOWLEDGEMENTS.

I wish to acknowledge the outstanding help and encouragement I received from the following persons in the preparation of this thesis. Without this kindly interest my task would have been immeasurably more difficult:

Professor K. J. MacGregor for his patience and guidance.

C. Nell, my honours partner, who helped initiate this project.

The staff at the Computer Centres of the University of Cape Town and the University of Natal, Pietermaritzburg for their aid and co-operation.

The Council for Scientific and Industrial Research and the Sperry Univac Corporation who provided generous financial assistance,

and finally all those who listened and criticized

Thank you sincerely.

ABSTRACT.

ADA is a high level programming language sponsored by the United States Department of Defence primarily for use in real-time systems. It has all the structures present in modern algorithmic languages with additional features for tasking.

This thesis discusses the University of Cape Town implementation of a front end processor for a subset of ADA. A compiler generator package was used to construct a syntax checker for the ADA language and a subset of this was extended through the semantic analysis phase finally to produce the intermediate code - DIANA.

DIANA is the standard intermediate code for all ADA programs and a representation for transfer between systems has been defined. DIANA is intended to function as an interface between the front and back ends of ADA compilers, and as an intermediate form which can be used by tools designed for ADA.

Table of contents

Table of contents

Table of contents

1. CHAPTER 1. INTRODUCTION.

ADA is a high level programming language initiated by the United States Department of Defence. During its development considerable effort went into the determination of the requirements of a language intended primarily for embedded computer applications. The resulting language is suitable not only for embedded computer applications, but also for general systems applications, numeric computation, real-time industrial applications, general applications programming, and for teaching good programming practices.

The first specification of the language became available in 1979 and in 1980 students at the University of Cape Town decided to gain experience in this new language by attempting to develop an ADA front end processor.

In producing the front end processor, a compiler generator package, GSA 1100, was used. This package provides the capability of writing a lexical analyser (accepting as input, a modified Backus Naur grammar), and an LL(K) parser. It also enables error recovery and calls to semantic routines to be embedded in the parser specification. A description of the GSA 1100 package is contained in CHAPTER 2.

The version of ADA used by the initial project was that released in 1979. This was modified into ADA-80 which became available in 1980, and is the version used by the final project. The 1980 definition did not entirely resolve the ambiguities and complexities of the 1979 specification, and ADA-80 had to be further refined in 1982.

The intermediate code produced by the front end was DIANA, an abstract syntax tree. It was designed for ADA-80 and has been

updated to be compatible with ADA-82. CHAPTER 3 discusses the history and features of these two languages.

While implementing ADA using GSA 1100, certain problems were encountered. Most of these occured because the ADA specification is ambiguous and contains many constructs unsuited to an LL(K) parser. The ADA specification was therefore restructured to permit parsing using GSA 1100. CHAPTER 4 describes how the syntax checker was constructed, highlighting the problem areas encountered and their solution.

ADA has rigid typing rules and the language specification requires that the static semantic checking be done at compile time. This entails type checking, evaluation of numeric expressions, checking for non-null ranges etc. To achieve this the program symbol table must be referenced. CHAPTER 5 describes the construction of the symbol table, and the routines that reference it while performing the semantic checking.

The output of the front end processor system is an abstract syntax tree representation of the intermediate code DIANA. DIANA was produced jointly by designers at Carnegie-Mellon University and the University of Karlsruhe for the ADA language. A description of DIANA is contained in CHAPTER 6.

CHAPTER 7 contains examples run on the front end processor and the resulting error messages and code produced.

2. CHAPTER 2. STRUCTURE OF THE ADA COMPILER.

The University of Cape Town ADA front end processor is written for the UNIVAC 1100/81. The body of the project is written in PLUS, Programming Language for Univac Systems, [Sperry Univac 1] and interfaces with tables generated by a syntax analyser package, GSA 1100, [Sperry Univac 2] provided by UNIVAC. Driver routines and interfaces are written in Univac assembler.

Figure 2-1 shows the basic structure of the envisaged compiler.

The structure of the UCT-front end processor is divided into five phases:

Phases 1, 2 and 3 are accomplished with the aid of the GSA 1100 package. Using the output produced by this package a symbol table is built which stores all tokens and their attributes as they are defined and later modified. This information is then used in the production of DIANA and enables the semantic information to be included in the DIANA structure.

Phase 4, the static semantic analysis involves checking that the tokens are correctly used, and so uses the symbol table produced by phases 1 and 2. If necessary, additional information may be inserted in the symbol table. In the case of errors, the message table produced in phase 3 is used.

In the production of DIANA, phase 5, all the semantic information comes from the symbol table, while the knowledge of which node to generate, or to where a particular node attribute should be linked, comes from the parse table generated in phase 2.

In this way, all the phases making up the front end are interdependent, and use the information produced by each other.

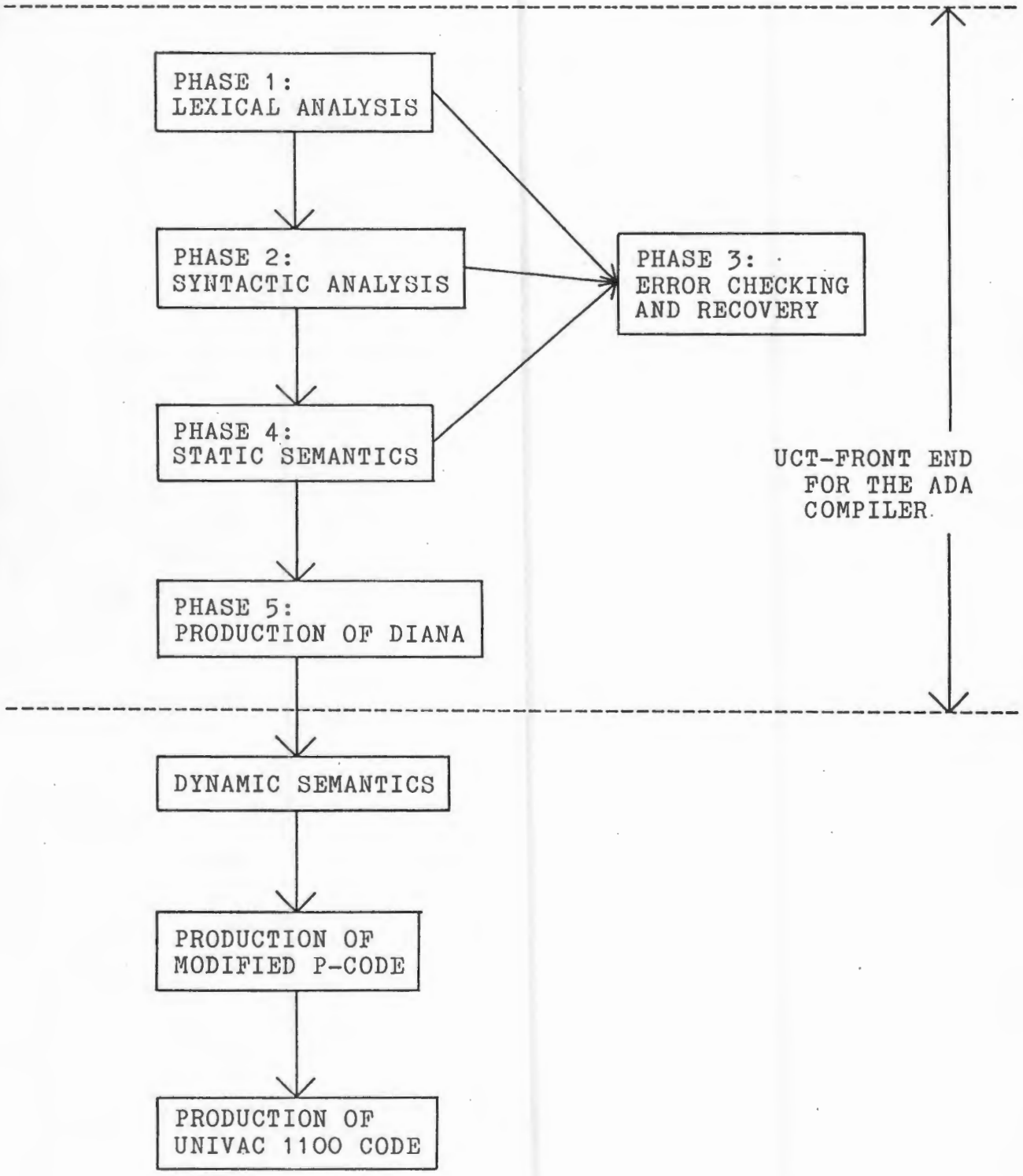


Figure 2-1. STRUCTURE OF THE UCT ADA COMPILER.

2.1. TABLE GENERATING ROUTINES.

The general syntax analyser consists of several table generating routines that take as input information about the language to be analysed, and produce as output various tables that may be interfaced with user-written routines. The table generating routines comprise the lexical analyser, parser, error message table and dictionary. Interfacing with each of the above is assembler code which manipulates the output produced by the package and passes tokens to user-written procedures via predefined stacks.

2.1.1. LEXICAL ANALYSER.

Input to the lexical analyser is in the form of a modified Backus Naur notation describing the composition of the terminal symbols that make up the lexical units of the language being defined. (See appendix I).

Once a lexical unit is recognised, it is transferred to the parser for parsing.

Each lexical unit is given a unique parameter number in the lexical specification, and this coincides with the same parameter number for the same token in the parser specification.

The lexical specification can be considered as a tree with many branches. It consists of only one production and so has one entry point, the root. Depending on the next character it follows a particular branch until a token is recognised, or an error found. Look-ahead should only be used in the lexical specification when absolutely necessary. This implementation had to use look-ahead to distinguish between a character literal e.g. 'R' and an

attribute reference e.g. A'FIRST. After recognising a single quote mark followed by a character, the path to follow depends on the next character in the input string.

2.1.2. PARSER.

Unlike the lexical analyser, the parser has many productions which make up the specification of the language. The same notation is used to input the rules of the language to the parser as is used to input the lexical units to the lexical analyser. Interspersed in the definition of the language may be calls to the error message routines and semantic routines. It is through these calls that the front end is constructed. The parameters at the beginning of the parser, lexical analyser and dictionary ensure that the same token has identical attributes in all the routines. The assembler routines link these parameters. Lexical units are defined and recognised in the lexical analyser and used in the parser. Other terminal symbols, e.g. reserved words, operators and special symbols may be defined in the dictionary and then used in particular places in the parser. All terminal symbols are listed at the beginning of the parser specification and given a unique parse number. This parse number is used by the semantic routines to identify these symbols. All non-terminal symbols used in the parser must define some production of the language and must also be listed at the beginning of the parser specification.
The general syntax analyser package is an LL(K) processor and allows look-ahead to any level in the parser. Look-ahead productions may be defined in the parser but this method was unsuccessful because difficulty was experienced when error processing and recovery was attempted. Look-ahead for terminal

symbols only was used.
There are various options made available by the parser to the compiler writer. One can mark the beginning of a production, inhibit stacking of certain symbols or reserved words, force a particular amount of unstacking say at the end of a production, and inhibit ambiguity checking.
ADA proved to be an ambiguous language and restructuring of productions, look-ahead in productions, semantic analysis and the use of the ambiguity stop had to be used in the specification of ADA for use with GSA 1100.

2.1.3. DICTIONARY.

The dictionary is a table containing all key words, symbols and operators. Through the dictionary one can give terminal symbols various attributes that may be used during semantics, e.g. giving order to the boolean values true, false. This routine sorts these terminal symbols into groups of characters for identification. It is important that the parse numbers of these key words is the same in all routines.

2.1.4. ERROR MESSAGE TABLE.

GSA 1100 gives the user an efficient way of constructing error messages and recovering to various parts in the syntax. A language and associated construction routines are provided to permit the production of meaningful error messages. As this software is part of the syntax analyser package, and not significant to the research being undertaken, it is not described. Again there is an assembler routine that interfaces with this

routine and the other table generating routines or user routines. It transfers the error token around and recovers to the correct places in the parser. An error recovery table is set up by the parser if there are any calls to the error message routine in the parser specification, and then the relevant error message is built as necessary. The call to the error message routine is done by referring to an error message number, or, in user-defined procedures, referencing the entry point of the error routine with the relevant error message number.
It can be seen that the above routines are very powerful and with their ancillary routines make an efficient tool in the designing and implementation of a compiler.

2.2. ANCILLARY ROUTINES.

Several other routines are required to interface with the table generating routines of the general syntax analyser package. These were largely supplied by UNIVAC and are responsible for initialisation of the system, searching and maintaining the stacks and interpreting the tabular output. A brief description of the most relevant one, token, follows.

2.2.1. TOKEN.

This routine is the most important of the supporting routines as it directs flow through all the routines. It is a user-written interface between the parser and lexical analyser and generates a token entry (detailed in CHAPTER 5) that is acceptable to the parser and any semantic routines that may reference it. While doing this it -

1) recognises key and reserved words by accessing the dictionary (the tables produced by the keyword builder).

2) maintains a dictionary of items recognised and so builds a symbol table.

3) converts and handles any special class of lexical units that map onto different token units. e.g. Certain words carry semantic information but the parser only needs these words to be recognised as identifiers.

Input to the token routine comes from the GSA 1100 lexical analyser. Output from the token routine consists of a single token entry that is placed in the input buffer of the parser.

The module containing the token routine has been extended to contain all the semantic checking. (See section 2.3 and CHAPTER 5).

The structure of GSA 1100 is modular, thus enabling the user to produce one error message table and one dictionary which may be used with all future releases of a language. Every time the language specification is changed, only the parser and lexical analyser need be updated. Emphasis can therefore be placed on implementing the language and not necessarily on error message production and recovery, which, once developed, remains static.

Figure 2-2 gives an overview of the way which the GSA 1100 table generating routines and interfacing routines work together to produce meaningful output.

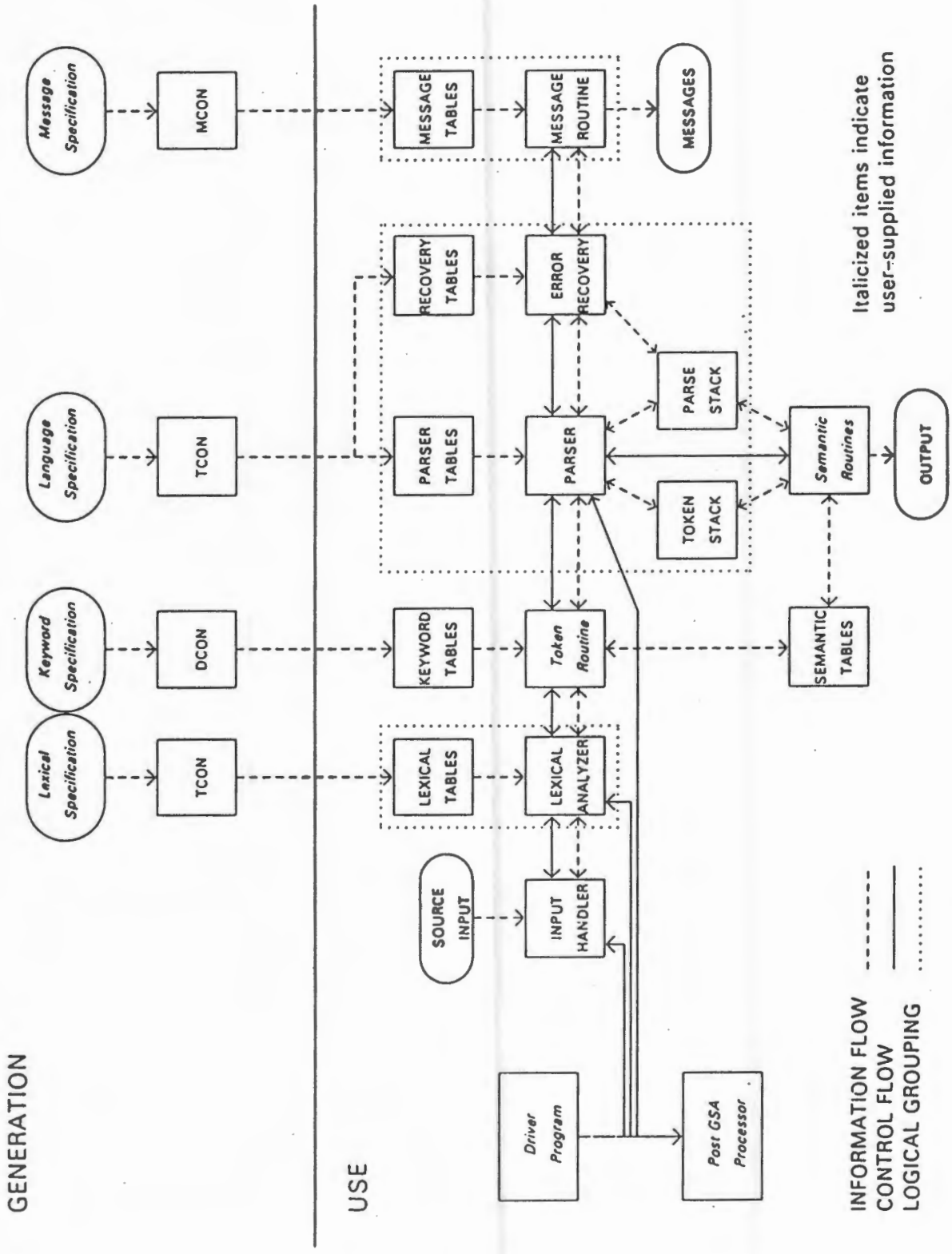


Figure 2-2. GSA 1100 SYSTEM OVERVIEW. [Sperry Univac 2]

2.3. STATIC SEMANTIC ANALYSIS.

Static semantic analysis is done through several procedures which reside in the module token. This was done because the token routine that interfaces with the GSA table generating routines produces the symbol table, token stack and various other arrays and stacks that are used in the semantic analysis. All the calls to these semantic routines are either imported from the parser or generated by other semantic checking routines.

Most of the static semantic checking is type checking which is achieved by using the information in the symbol table together with the token stack. Intermediate results are stored on the token stack and where static expressions are evaluated, the results are stored in an array of values and a pointer is set up from the symbol table entry to the expression value.

Overloading within numeric types and the converting of strings of numeric digits to their numeric value is catered for. Where possible, range checking in the form of type compatibility and non-null ranges is done, and each array subscript is tested for compatibility between it and the allowed subscript type for that array.

For a detailed description of these procedures, see CHAPTER 5.

The intermediate code being produced is DIANA, and where possible, the DIANA constructs include semantic information e.g. numeric values, intermediate types etc. Thus calls to the DIANA producing routine are made from the semantic analysen procedures where applicable.

The token module is therefore the pivot of the whole front end processor.

2.4. PRODUCTION OF DIANA.

Two of the original design goals of DIANA were:

1) " DIANA is based on the Formal Definition of ADA " and

2) " in DIANA, there is a single definition of each ADA entity " [Goos 1981].

These two goals conformed to the method of construction of the UCT-front end and resulted in calls to the DIANA producing routine primarily being made from the parser. As each construct is identified in the parser, the relevant DIANA code is produced with any associated semantics from the symbol table.

It is against the batch processing philosophy to produce code for an incorrect program. Thus, once a compilation error is found, no more DIANA is produced. This allows the production of DIANA to assume a syntactically correct ADA program as input.

Another of the design goals was uniformity within the DIANA constructs. Consequently different DIANA structures may have similar configurations which rely on scope and context sensitive information for their construction. Scope problems were overcome by using several stacks keeping track of the position of certain nodes in the DIANA tree, block numbers, beginning of lists, etc. Once a construct is complete, it is output to a file.

As each DIANA node is generated, the DIANA routine attempts to complete it. Every DIANA node has the same basic structure and may be interpreted through a unique identifying number. By using this number and correlating it with the kind of node, one can determine what the subsequent attributes mean.

The production of this intermediate code is performed in the module DIANA which is written in PLUS. For a detailed description see CHAPTER 6.

3. CHAPTER 3. BACKGROUND INFORMATION.

3.1. ADA.

3.1.1. HISTORY.

In the early 1970's the United States Department of Defence decided that it was necessary to do something about the rising software costs; the most important of which was that of the embedded system sector (approximately $1680 million). To counter this it was decided that a single standard was needed for this sector.

A set of requirements was developed and refined in a series of documents.

DATE	DOCUMENT	PURPOSE
1975	Strawman	Informal comments
1975	Woodenman	Informal comments
Jan 1976	Tinman	Proposed language
Jan 1977	Ironman	Informal comments
April 1977		Call for tender
July 1977	Revised Ironman	
	PHASE 1 GREEN RED BLUE YELLOW	
	Steelman	
April 1978	PHASE 2 GREEN RED	
	Preliminary ADA	
May 1979	PHASE 3 GREEN	
July 1980	ADA-80	
July 1982	Draft ANSI standard	

The call to tender for the language in April 1977 received seventeen replies. In August 1977, when Phase 1 began, four of these were chosen. Each was designated a colour code: CII-HONEYWELL-BULL (green), INTERMETRICS (red), SOFTECH (blue), and SRI INTERNATIONAL (yellow). In May 1979 the green language was accepted. This was designated Preliminary ADA.

After much criticism and experimentation, changes were made to preliminary ADA [Preliminary 1979] resulting in ADA-80 [Military 1982]. ADA-80 overcame many of the ambiguities and semantic problems that existed in the preliminary language.

ADA-80 was subsequently reviewed and modified in the light of implementation difficulties and a draft standard document was released [Reference 1982]. This document became available to the public in October 1982 and appears to have resolved and clarified many of the hazy areas of the previous two releases of the language.

ADA compilers fulfilling the validation requirement of the Department of Defence [Goodenough 1981] are being produced internationally. Of necessity these compilers' abilities include the specified tasking capabilities of the language. All leading manufacturers and several large institutions are currently supporting, sponsoring and implementing ADA. This should result in ADA becoming the real-time standard of the 80's.

An important complementary activity was the development of a parallel series of documents entitled SANDMAN, PEBBLEMAN and STONEMAN as the ADA support environment [Stenning 1981, Wolfe 1981] . This enhances program portability and allows programmers' proficiency in one standard language.

3.1.2. THE LANGUAGE.

ADA is the programming language designed according to the STEELMAN requirements of the United States Department of Defence [Steelman 1978] . ADA was named after Ada Lovelace who was a leading computer pioneer of the nineteenth century, a colleague of Charles Babbage, and the daughter of Lord Byron. It was designed at CII-HONEYWELL-BULL by a Paris based team led by Jean Icbiah.

The main basis of ADA is the programming language PASCAL and its later derivatives. A goal in the design of ADA was to retain the Pascal spirit of simplicity, but not necessarily the form of each Pascal feature.

ADA possesses the data abstraction features of modern programming languages, and as such, has the ability to define types and subprograms with the conventional control structures serving the need for modularity. An integral part of the language is real-time programming with the ability to model parallel processing and cater for exception handling. Included are systems program applications requiring access to system dependent parameters and precise control over the representation of data.

The three design criteria for ADA were

1) the importance of program reliability and maintenance.
2) the fact that programming is a human activity.
3) programs are required to be efficient.

The designers endeavoured to make it easy for programmers to write good, structured programs using ADA.

An attempt was made to keep the language as contained as possible. This gave rise to a language with a minimum number of underlying concepts put together in a consistent and ordered manner.

A program in ADA is a sequence of higher level program units.

Program units may be subprograms, package modules or task modules. Separate compilation of program units allows a program to be designed, written and tested in independent parts.
A subprogram may be either a procedure or function subprogram, and is the basic unit for expressing an algorithm. A procedure subprogram performs a sequence of desired actions, whereas a function subprogram computes a value. A package module defines a collection of logically related units. Portions of packages can be hidden from the user, allowing access only to the logical properties of the module. A package module may be a collection of related subprograms.
A task module also defines collections of logically related units but with the additional capabilities of parallel processing. Tasks may be implemented on multiple processors, or interleaved on a single processor.

3.1.3. VERSIONS OF ADA.

Preliminary ADA was released in 1979 [Preliminary 1979] and had some very interesting features, e.g. dynamic arrays, initial default values for parameters, the idea of separate compilation units, etc. It left many areas unexplained and ambiguous. This version was studied and discussed by people throughout the world and when ADA-80 was released there were significant changes. For a detailed list see [Winkler 1981].
One noticeable difference between the two documents was that the ADA-80 specification was more detailed and more precise than Preliminary ADA. In several instances restrictions and details were specified where previously no elaboration had existed.
The UCT-front end is based on the ADA-80 definition of the

language [Military 1980] where there are still several flaws with which implementors have to contend. The draft standard document for ADA [Reference 1982] overcomes several of the problems present in ADA-80. Below are a few of the syntactical changes between ADA-80 and "final" ADA. Only changes relevant to the UCT-front end are mentioned.

The new manual explicitly details the semantics and rules of the language including notes to the implementor about the interpretation of the explanations.

The syntactical differences included in this implementation are:

1) NAME has been restructured. The UCT-front end has taken the final ADA definition of SELECTOR.

2) The relation part of an EXPRESSION has been simplified and this simplification has been implemented.

3) Wherever an EXCEPTION_HANDLER appeared in ADA-80, it was optional, even if the reserved word EXCEPTION was present. This has been changed. If the word EXCEPTION is used, it must be followed by at least one EXCEPTION_HANDLER. This has also been implemented in the UCT-front end.

Other changes are:

1) DECLARATION has been changed to overcome an ambiguity that existed in the syntax for DECLARATIVE_PART. Several productions that may have appeared further down the parse tree have been moved so that the correct path can be identified as early as possible. Consequently DECLARATIVE_PART and all the productions it calls have been redefined.

2) An OBJECT_DECLARATION can no longer contain an unconstrained ARRAY_TYPE_DEFINITION. This infers that all unconstrained array types must be explicitly declared as types.

3) The syntax for TYPE_DECLARATION has been altered to include a

PRIVATE_TYPE_DECLARATION. This has no actual significance in the structure of a TYPE_DECLARATION as the PRIVATE_TYPE_DEFINITION has been taken out of TYPE_DEFINITION causing no difference to the resulting syntax.

4) RANGE has been changed to include the range attribute.

5) In several places where previously a SUBTYPE_INDICATION was specified, ADA now only requires a TYPE_MARK. Also where NAME was specified, ADA now requires only a SIMPLE_NAME which is an IDENTIFIER. These two simplifications make the whole syntax less ambiguous and easier to implement.

6) The components of a record may no longer be declarations of arrays but just variables of an existing user-defined or standard type. Also, a COMPONENT_LIST must now have at least one component, even if it is the NULL component.

7) ALLOCATOR has been simplified.

8) One of the major differences is that of a parameterless function call. In ADA-80, an empty pair of parentheses had to follow the call. This is no longer required.

The remaining syntactic changes refer to packages, tasks, generics and library units, all of which have not been implemented, so are not detailed here. They may be found in [Harrison 1982].

The changes made to "final" ADA are mainly for the better as they bring ADA in line with other current languages. Although programmers are accustomed to the change in FUNCTION_CALL, this will continue to cause further ambiguities for the implementor.

3.1.4. RESTRICTIONS.

As the implementation of a compiler for the complete ADA specification would involve tens of man years work, only a subset could be implemented. The syntax checker has been constructed for the complete ADA definition (except the RENAMES clause), but only a subset has been extended further.

The following rationale was used to extract the subset. A program in ADA is a sequence of higher level program units. Program units may be subprograms, package modules or task modules.

The basic unit for expressing an algorithm is a subprogram which may be split into procedural subprograms and function subprograms. A package module defines a collection of logically related entities. A task module is basically a package module with additional capabilities for parallel processing. Since modules require the basic constructs of a subprogram, it was decided to implement only subprograms. This subset is equivalent to PASCAL. Just as PASCAL can easily be extended to concurrent PASCAL, the selected subset can be extended to encompass the whole of ADA using GSA 1100.

In particular the following were not implemented: pragmas, modules, visibility rules, tasks, program structures and compilation issues, generic program units, representation specifications, implementation dependent features and the RENAMES clause.

Pragmas are messages to the compiler. They were excluded because insufficient information was known about GSA 1100 to be able to implement pragmas successfully.

Task modules and package modules were excluded since they are not really applicable in the present environment. Generally, students

do not work in groups on a single package, and therefore the need for packages is small. On a single processor system tasks are usually only useful for simulation, and therefore they were excluded.

Since the visibility rules are intended for modules, they were removed from the workable subset. Compilation issues are used mainly for packages in program libraries, and are not relevant to a subset excluding separate compilation and modules.

Other parts were originally excluded from the subset, but were later inserted as their meaning became clearer and they were seen to be part of the basic subset. This subset has been implemented as far as the production of DIANA, that is syntax checking, static semantic checking and the production of DIANA. As the remainder of ADA has been implemented in the syntax checker, this implementation can be extended through the semantic phase to the production of DIANA to give a complete front end for ADA.

3.2. DIANA.

3.2.1. INTRODUCTION.

DIANA, a Descriptive Intermediate Attribute Notation for ADA is based on two other intermediate codes - TCOL and AIDA. DIANA is expected to become the standard intermediate language for ADA implementations, and so we decided to use it as the end point for the UCT-front end.

The design of the language, as stated in the DIANA manual, [Goos 1981] is based on the following principles:

1) It is representation independent.

2) It is based on the formal definition of ADA.

3) Regularity is a principal characteristic of DIANA.

4) It must be efficiently implementable.

5) Consideration of the kinds of processing to be done is paramount.

6) In DIANA, there is a single definition of each ADA entity.

7) DIANA must respond to the issues posed by ADA's separate compilation facility.

8) There must be at least one form of DIANA representation that can be communicated between computing systems.

These design goals give rise to an intermediate language that has a syntax similar to ADA produces uniform structures for the different productions and is easy to implement as the user sees fit. The resulting intermediate form of the program resembles the original program in such a way that there should be no problem of regenerating the source, or of designing tools which use the "DIANA" program.

Since the original release of DIANA (March 1981) several problems have been encountered. These have been studied and a new draft release of DIANA (October 1982) [Evans 1982] attempts to overcome these. DIANA-82 is compatible with the draft proposed ANSI standard for ADA.

As the UCT-front end is based on ADA-80, it used DIANA-81.

3.2.2. THE LANGUAGE.

An instance of the DIANA form of an ADA program is an attributed tree. The structure of the tree is that of the abstract syntax tree defined in the ADA formal definition. Attributes of the nodes of this tree encode the results of semantic analysis.

The DIANA tree is structured according to the syntax of ADA. Each ADA production has a corresponding DIANA production giving rise to a set of nodes for that production. Any static semantic analysis is held in these nodes through the semantic attributes of the node. If that node relies on system supplied information, there will be attributes for this information specified in this node. In this manner, the DIANA tree contains the results of the syntax analysis, static semantic analysis and system supplied information.

Every node has the attribute LX-COMMENTS and most have the attribute LX-SRCPOS. The latter is the source position of the symbol and these two attributes are for use in regenerating the source code.

```
ADA SYNTAX    : ASSIGNMENT_STM ::= VAR_NAME := EXPRESSION

DIANA SYNTAX : ASSIGN => as-name      : NAME
                         as-exp       : EXP

               ASSIGN => lx-srcpos    : SOURCE POSITION
                         lx-comments  : COMMENTS .
```

EXAMPLE A := 3

The DIANA tree to be produced would be

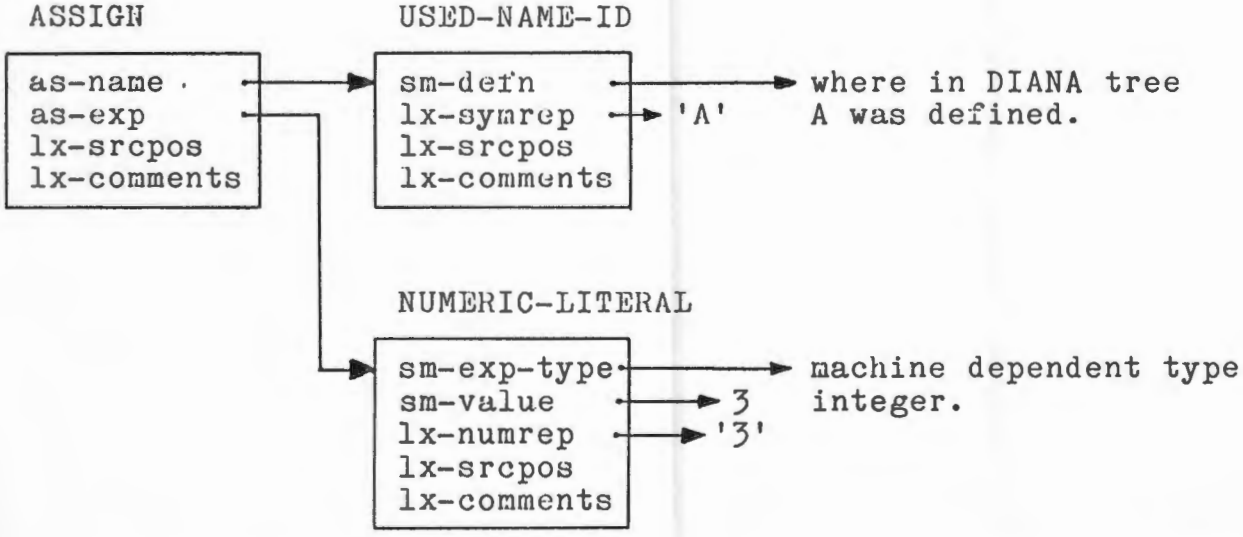


Figure 3-1. EXAMPLE OF AN ADA STATEMENT AND ITS DIANA TREE.

Figure 3-1 is an example of an ADA statement and its equivalent DIANA tree. The section of tree produced would be linked to the other statements in the block of statements, or to the block header.

DIANA has proved easy to implement, especially due to the almost one to one mapping between each ADA production and each DIANA tree.

4. CHAPTER 4. THE SYNTAX CHECKER.

Four modules constitute the Syntax Checker. They are the lexical analyser, parser, dictionary and error message table. The pivot of these is the parser. It uses information from the other three modules to do the syntax checking specified.

CHAPTER 2 briefly describes the function of these modules and this chapter will only discuss how ADA has been implemented using the facilities of the GSA 1100 parser.

The parser is a top-down, look-ahead parser that uses the tables built from the language specifications to control parsing. It can be envisaged as an interpreter that has a repertoire consisting of recognition, control and action instructions.

The token routine passes input recognised in the lexical analyser to the parser through a "window". Depending on the input and the present state of the parser, the appropriate action is taken. As input is recognised it is placed on the token stack available for semantic routines.

Look-ahead is used to resolve ambiguities by specifying look-ahead at terminal symbols in the input stream. It is specified and interpreted as follows:

<STATEMENT> ::= <NAME> ('(' <PARAMETER_LIST> ')')
?[':=' : ':=' <EXPRESSION>]? ';' .

This is read as :

1) Accept a <NAME> .

2) If the next token is a '(' accept '(' <PARAMETER_LIST> ')'.

3) Look ahead at the next symbol. If it is a ':=' then accept ':=' <EXPRESSION> and proceed to 4). If it is not ':=' then proceed to 4.

4) Accept a ';'.

Sets are used to specify the recognition of any set of terminal symbols e.g. all the names that could be parsed as identifiers.
Interspersed between the productions making up the ADA language are calls to the error message routines indicating which error message to print and what recovery is needed, calls to the token module routines controlling semantic checking, and calls to the DIANA module directing the production of the DIANA code.
In implementing a syntax checker for ADA, the semantics of the language were ignored and only the syntax considered. This was found to be ambiguous and overspecified. Due to the structure of the GSA 1100 parser, these problems became the most difficult to solve of those encountered.
Ambiguity means that at a certain point in the syntax, it is not clear from the input item which path in the syntax is being followed. Overspecification is when alternate paths in the syntax cannot be distinguished from one another. Overspecification is the worst case of ambiguity.

4.1. AMBIGUITY RESOLUTION.

An example of the kind of problem encountered can be seen in the condensed version of the following two productions (ADA manual sections 3.7.3, 4.3 [Military 1980])

<CHOICE> ::= <SIMPLE_EXPRESSION> \ <DISCRETE_RANGE> \ OTHERS .
<COMPONENT_ASSOCIATION> ::= (<CHOICE> ('|' <CHOICE>) ... =>)
<EXPRESSION> .

Suppose a token is recognised as:

1) IDENTIFIER. This may be parsed as a <DISCRETE_RANGE>, <SIMPLE_ EXPRESSION> or <EXPRESSION> .

2) NAME. The same three options exist.

3) SIMPLE_EXPRESSION. The same three options exist.

It can be seen that even after recognising a complex structure such as <SIMPLE_EXPRESSION>, there is no way of knowing which path to follow.

This ambiguity does not exist only if an identifier is found, but also if a number, CHARACTER_STRING, ALLOCATOR or left parenthesis is found. For example, say a '(' is found. The next token may be an expression of any complexity followed by the closing parenthesis ')'. The sequence "(EXPRESSION)" can be a parenthesised EXPRESSION or an AGGREGATE. Both of these occur as a PRIMARY. This may be recognised as <SIMPLE_EXPRESSION>, <DISCRETE_RANGE> or an <EXPRESSION>.

Ambiguities occur frequently, and where one occurs, it may involve many productions due to the design goal of a minimum number of underlying concepts. They are resolved in one of three ways, restructuring of the productions, through the use of look-ahead and with the aid of semantic information.

Restructuring involves combining the ambiguous parts of each production to create a new production used by the previous two ambiguous productions.

This technique may be used in conjuction with the look-ahead technique to resolve the ambiguity completely. (see example, figure 4-1). Look-ahead is done only for terminal symbols. This may require specifying several terminal symbols before reaching one that will identify which path to follow.

In the following example, restructuring of the syntax and look-ahead have been used to resolve this simple ambiguity.

Figure 4-1. EXAMPLE SHOWING AMBIGUITY RESOLUTION BY RESTRUCTURING

```
<DECLARATION> ::= [ <TYPE_DECLARATION> \ ....
                  \ <OBJECT_DECLARATION>
                  \ <EXCEPTION_DECLARATION>
                  \ <NUMBER_DECLARATION> ] .

<OBJECT_DECLARATION> ::= <IDENTIFIER_LIST> ':' ( CONSTANT )
                         <SUBTYPE_INDICATION>
                         ( ':=' <EXPRESSION> ) ';' .

<NUMBER_DECLARATION> ::= <IDENTIFIER_LIST> ':' CONSTANT
                         ':=' <EXPRESSION> ';' .

<EXCEPTION_DECLARATION> ::= <IDENTIFIER_LIST> ':' EXCEPTION ';' .
```

The ambiguity in the last three productions is clear, and can be resolved only after the colon. An ambiguity still exists between <OBJECT_DECLARATION> and <NUMBER_DECLARATION>, so look-ahead was used to look for the assignment sign which identifies the number declaration. The resulting declarations are:

```
<DECLARATION> ::= [ <TYPE_DECLARATION> \ ....
                   \ <IDENTIFIER_LIST> ':'
                  [ <EXCEPTION_DECLARATION> \
                  ?[ CONSTANT ':=' : <NUMBER_DECLARATION> -> Q ]?
                  <OBJECT_DECLARATION> ] ] ^ Q .

<OBJECT_DECLARATION> ::= ( CONSTANT )
                         <SUBTYPE_INDICATION>
                         ( ':=' <EXPRESSION> ) ';' .

<NUMBER_DECLARATION> ::= CONSTANT
                         ':=' <EXPRESSION> ';' .
```

<EXCEPTION_DECLARATION> ::= EXCEPTION ';' .

A common cause of ambiguities was the use of NAME. NAME starts with an IDENTIFIER, where an IDENTIFIER is any sequence of letters, digits and underscores. Often the type of IDENTIFIER used was context dependent and this was used in solving the ambiguities by semantics.

Every IDENTIFIER used in a DECLARATION was flagged as having a particular characteristic, e.g. variable identifiers, type identifiers, constant identifiers etc. Where it could be determined that a particular context permitted only a particular type of identifier, this is all that was allowed. This idea worked very well but became too restrictive and began trapping semantic errors before the syntax had been checked. As a result, semantic checking of this sort is used with care.

Another kind of semantic checking used is where the path to take is determined by characteristics of the token just parsed. This method solved the terrible ambiguity of NAME. An array reference has a subscript list which is all part of NAME and a PROCEDURE_CALL may have a PARAMETER_LIST. With a PROCEDURE_CALL only the procedure NAME is parsed as part of NAME.

Oversimplified, both start with a NAME optionally followed by '(' IDENTIFIER. The ambiguity is when the open parenthesis is recognised. Is it part of NAME or part of a PARAMETER_LIST?. This was resolved by checking to see if the NAME before it is an array NAME. If so, the whole construct to the next close parenthesis is parsed as NAME otherwise the NAME ends before the open parenthesis.

In the article "Ada Syntax Diagrams for Top-Down Analysis" [Bonet], R. BONET et AL discuss how they reduced the ADA syntax to be LL(1). The methods used are very similar to those used by the

UCT implementation. Their restructuring appears to be more ordered and they make greater use of semantic information to achieve their goal.

The UCT syntax checker implements the whole of ADA (excluding the RENAMES clause and separate compilation) and does full error checking with recovery where possible. There are areas which could be improved and the ambiguities resolved in a more efficient manner. This was not done as the next phase in the development of the front end appeared to be more challenging.

5. CHAPTER 5. THE TOKEN MODULE.

Static semantic checking constitutes all checking that can be done at compile time. The procedures that do this checking are all found in MODULE TOKEN and use the tokens on the token stack together with their symbol table information. Results of this checking may be temporarily stored on the token stack for immediate re-use or stored in various other stacks and arrays linked to the symbol table.

5.1. STRUCTURE OF THE SYMBOL TABLE.

The symbol table is a structure consisting of linked lists of nodes, where each list header is accessible through a hashing function. Each node is a symbol table entry and may be referenced through a pointer to the symbol table. Symbol table entries take the form of a PLUS (Programming Language for Univac Systems) record, each field holding different semantic information or references to other structures holding information.
A symbol table entry occupies seven words of storage. These are divided into the fields illustrated in figure 5-1 which hold the following information:

5.1.1. STRUCTURE.

The structure of a token is the number of characters and words making up the token, its lexical level, block number and lexical type. Once a token has been identified by the lexical analyser, these values are known or calculated and stored for further reference.

BITS WORD	0 3 6 9 12 15 18 21 24 27 30 33 36									
1	PARSE TYPE	NUM. CHARS	LEX TYPE	CLASS VALUE	NUM. WORDS					
2	CLASS	LEX LEVEL	NEW ENTRY	BLOCK NUM.						
3	POINTER: NEXT-DICT		POINTER: ITEM-LINK							
4	BASE TYPE	COMP TYPE	PARSE DEF	INDEX	INCOMPL	DIMENSIONS	DIM LINK	DERIVED	DYNARRY	
5	POINTER: TYPE-PTR		POINTER: NEXT-VAR							
6	SM-DEFN		BSETPE							
7	TPE-STRUCT		UNUSED							

WORDS 1-5 CONTAIN INFORMATION FOR STATIC SEMANTIC CHECKING.
WORDS 6-7 CONTAIN SEMANTIC INFORMATION USED BY THE DIANA MODULE.

Figure 5-1. STRUCTURE OF A SYMBOL TABLE ENTRY.

5.1.2. SEMANTIC INFORMATION.

Semantic information is divided into two catagories, the information used in the static semantic checking and the information necessary to complete the semantic pointers of DIANA nodes.

Information used for static semantics includes:

1) PARSE-TYPE. A number used to identify a particular group of terminal symbols used in the parser. e.g. If an identifier is

defining a new type, the identifier will have a parse-type indicating a type-name.

2) PARSE-DEF. A number used to identify the type definition being defined. e.g. Record, array.

3) CLASS-VALUE. Each predefined type has a unique type number for identification. Each user-defined type must also be given a unique type number. These are stored in the CLASS-VALUE field. This field is also used to indicate whether a variable is defined to be a constant.

4) CLASS. Within each CLASS-VALUE there may be several unique groupings to be identified. e.g. The type BOOLEAN has two values, TRUE and FALSE. They are defined to have a particular order and this is held in this field.

The values for the PARSE-TYPE, CLASS-VALUE, and CLASS may be pre-set in the dictionary.

5) BASE-TYPE. The type of the first declaration in a string of declarations constitutes the base-type of a variable. Overloading may require that the base-type of a definition be used. This could be found by backtracking through the connecting pointers. The structure of the first declaration is useful in determining properties of a variable and this could also be obtained through backtracking. If only backtracking were used the presence or absence of the certain properties would be recalculated each time a particular variable was used. Therefore two extra fields holding the BASE-TYPE and component type (COMP-TYPE) of declarations are included in each entry. The component type is only meaningful when a complex structure e.g. array or record is being defined.

6) Various other fields indicate whether or not a type declaration is complete or derived and if the entry in the symbol

table is a new entry. If it is, certain semantic information must be supplied and the new entry information is used by a routine called from the parser.

7) In each entry for an array variable is a field holding the number of dimensions of the array.

5.1.3. POINTERS.

Several tokens may constitute a complex structure, e.g. record. The elements of this structure are linked together by Pointers. Pointers are also used to link variables to their type definitions.

Numeric values and the results of expression evaluation are stored in arrays. The value of the indices to these various arrays are stored in particular fields in each entry. Arrays may have indices of different types. These types are stored in a type array and the position in this array is specific to a particular definition, and is stored in the symbol table entry.

5.1.4. REPRESENTATION.

At the end of every entry in the symbol table, the ASCII representation of the token is stored. The number of words needed to do this is calculated by the routine token and is stored in the symbol table for future use.

DIANA symbol table information is completed and referenced in the DIANA module.

Figure 5-2 shows the basic symbol table entry showing linkage for the following extract of ADA.

```
TYPE ONE IS RECORD
 D,E : INTEGER ;
 F   : FLOAT   ;
END RECORD ;
TWO : ONE ;
```

Figure 5-2. SYMBOL TABLE LINKAGE.

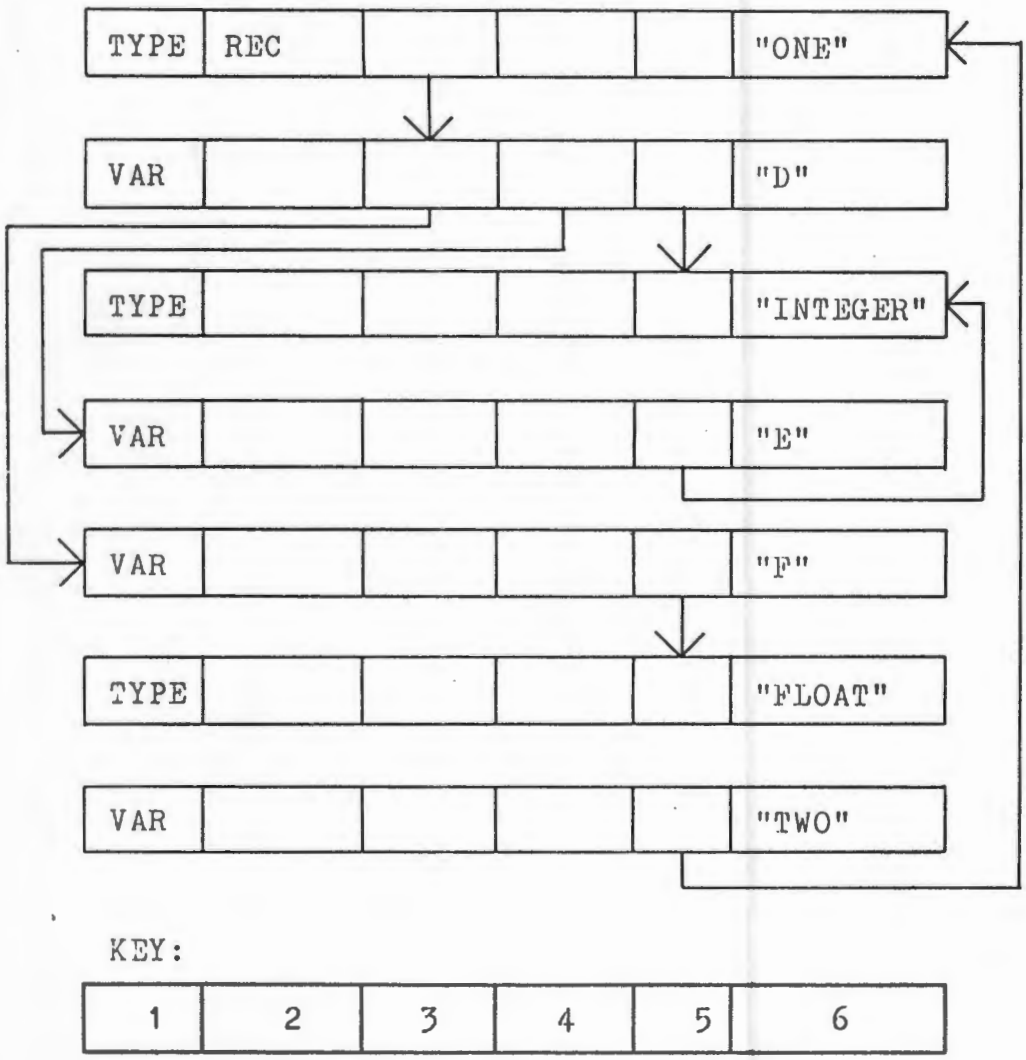


figure 5-2. CONTINUED.

1: PARSE-TYPE of entry.

2: PARSE-DEF of entry.

3: Pointer linking the record definition to the first element of every list in the component list.

4: Pointer linking elements making up each list in the component list.

5: Type pointer linking the variable to its type definition.

6: Representation.

5.2. FUNCTION OF THE TOKEN MODULE.

5.2.1. PRE-PARSER PROCESSING.

Lexical units are constructed by the lexical analyser and passed to the token procedure of the token module with the number of characters constituting the unit. The token procedure uses the number of characters to calculate the number of words the symbol will occupy, and the hashing function value needed to access the symbol table. It then searches through the symbol table, using the hashing information to establish if the symbol has been entered before. If not, a new entry is created. This new entry contains the symbol representation, parse and lexical types (which remain the same until the parse-type is modified during post-parser processing), number of characters and number of words making up the symbol, and a flag which is set to indicate that a new entry has been created. Once the entry has been created a check is made to see if it is an identifier or a special symbol. If it is an identifier, it could be a reserved word which has

special properties defined in the dictionary.
The results of this check, direct the token procedure to return either the token to be passed to the parser, or to search the dictionary for additional information before returning the token. The dictionary supplies the token entry of specified tokens with the parse-type, class and class-value. If the tokens are not specified in the dictionary, these fields will be completed during post-parser processing.
The search of the symbol table will often result in a match. This may or may not be a legal occurrence of the token so the token, with its semantic information is returned to the parser for post-parser processing.

5.2.2. POST-PARSER PROCESSING.

DECLARATIONS.

CHAPTER 4 refers to using semantic information to solve ambiguities in the ADA language. The semantic information used is that obtained from the declarations of the variables and their types.
ADA defines several forms of named entities, the most important being TYPE, NUMBER and OBJECT declarations. Each TYPE_DECLARATION defines a particular structure e.g. record, array, access type, etc. Each DECLARATION creates an IDENTIFIER with a special function and properties, and perhaps structure. This information must be encoded into the symbol table, and this is done through calls from the parser.
Every declarative production initiates a call to a "declaration" procedure in the token module. This procedure accepts the token created or found during pre-parser processing. If the new entry

flag is true, it is an IDENTIFIER that has not been declared before, and the PARSE-TYPE field can be completed without any further processing. This routine may encounter identifiers that have been declared before, either at an earlier lexical level or in error. If the new entry flag is false, the lexical level of the symbol table entry and the current lexical level are compared. An error has occured if the levels are the same, but if they are different a new entry is created for the IDENTIFIER.

The entry for the IDENTIFIER is marked with a pointer to enable it to be completed as more semantic information pertaining to its declaration becomes known. A call from the parser would indicate the type of structure being defined and this is entered in the PARSE-def field of the entry.

The pointers are all created from one entry to another as a result of calls from the parser.

Every time the symbol is referenced in a production that is not a declaration, the above information is available for use in semantic checking. Figure 5-2 indicates that this semantic information constitutes the most important fields in the symbol table. The remaining fields are completed during the parsing of the body of the program.

SCOPE.

Scope is an important factor in any computer language. Where the same variable is being defined twice, the symbol table relies on lexical levels to determine whether the definition is a legal redefinition of an inner level, or an error.

The token module houses routines that increment and decrement block numbers and lexical levels at the appropriate times.

INITIALISATION.

The token routine performs certain initialising routines.
It assigns initial values and properties to the predefined types e.g. INTEGER, FLOAT, BOOLEAN etc. They have been defined to have a lexical level of zero and may be redefined by the user at a higher lexical level. Thus each of these types is an entry in the symbol table and may be referred to just as any other entry. Any predefined entry that may be redefined must be initialised in token and not in the dictionary because the dictionary only allows one definition of each key word, whereas the token routine is structured to allow redefinition of identifiers at different lexical levels.
Flags, arrays and counters are explicitly initialised when they are declared, or in an initialising procedure.

STRUCTURES AND STACKS USED BY THE TOKEN MODULE.

1) The token stack is a stack created by the parser. Tokens of a production are loaded on to this stack and are accessed by the semantic routines. Each token on the stack may be linked to its symbol table entry to enable the semantic routines to access the information of the token. Using this stack, each token entry can be traversed and evaluated or inspected. A pointer indicates the current position in the stack and may be moved along the stack for whatever processing is required. The implementor may add entries to this stack for semantic use and it is through this stack and its pointers that most of the semantic checking is performed and transferred to the symbol table.
2) Each declaration of an array may have several indices, each of which may have different types. These types are stored in an array. The number of dimensions of the array declaration is

stored in the symbol table entry for that array and its number of indices is therefore known. The index to the type-array which refers to the type of the first index is stored in the symbol table and by using the number of dimensions field in the symbol table the others may be located for subscript checking. Used in conjunction with this type-array is a bound-array. It is used not only for array variables but for any variables with an associated range. If the range is static, the bounds of the range are stored in this bound-array where they are accessed for range checking. A similar mechanism to that of the type-array index is used for the bound-array.

3) Numeric strings are converted to their equivalent numeric value and stored in one of two arrays. There is an array for storing reals and one for integers. Once a string has been converted, its value is linked to the symbol table entry. The next identical string recognised need not be converted to its numeric equivalent as this has been stored.

4) Type compatibility is a major part of the static semantic checking. A complex structure may not necessarily be used in its most complex form, thus type checking does not involve using the base type of a structure. Often the type of an intermediate structure is temporarily required and after some processing, the type of the next stage of the structure is required. An array of pointers is used to keep particular positions in the structure for further reference. This array works as a stack, increasing and decreasing with complexity of the structure.

EXAMPLE.

```
TYPE R IS ARRAY(1 .. 10) OF INTEGER ;
TYPE S IS ARRAY(10.. 20) OF INTEGER ;
TYPE S1 IS ARRAY(10.. 20) OF FLOAT   ;
TYPE U IS ARRAY(1 .. 10) OF S1   ;
I : INTEGER ;
B : S ;
A : R ;
D : U ;
.
.
.
D(B(I))(A(I)) := 3.4
```

Sequence of events.

1) Find the type of D( ).

2) Check subsript type B(I). This entails leaving the current chain of backtracking to ascertain the type of B( ) and then checking the index I. Once B(I) has been verified against the subscript type of U, control must return to the chain of backtracking which reached D( ) to continue with D( ) ( ). This is done by storing the symbol table entry pointed to by the backtracking pointer in the array of pointers. This method continues until the type of the structure has been reached.

Using the same declarations, and introducing the declaration

```
E : S1 ;
```

D(3) := E ; is legal.

This is an instance when the sequence of backtracking events does not need to go all the way to the base type.

5) Static semantics involve evaluating static expressions. The operands of an expression are not evaluated in a sequential manner. Intermediate results, and types needed for re-use are stored using two stacks, one for the values and one for the types. The stack with the intermediate types also indicates whether the result of an evaluation is a constant. This is used in overloading resolution of arithmetic expressions.
Several other stacks are used in particular procedures to perform very specific tasks.

OVERLOADING.

The UCT-front end does not yet cater for overloading of the form:

```
TYPE COLOUR IS (RED,GREEN,YELLOW,BLUE,ORANGE) ;
TYPE RAINBOW IS (RED,ORANGE,GREEN) ;
VARCOLOUR : COLOUR ;
VARRAINBOW : RAINBOW ;
.
.
.
VARCOLOUR := RED ;
```

The last statement may cause an error as the type checking routine may identify RED as being of type RAINBOW instead of COLOUR. This is because the token procedure which sets up the symbol table allows only one occurrence of an identifier at any particular lexical level. In the above example there are two occurrences of RED, ORANGE, and GREEN and an error would have been produced when an attempt was made to parse the type declarations.
This characteristic gave rise to several problems. The first is that the ADA language specification insists that overloading be allowed and the UCT-front end does not allow it. The second is

that records are an integral part of the language. Not to be able to use the same name for fields in two different records seemed a limitation to the user. The later problem was solved in the following manner.

Consider the declarations

```
TYPE A1 IS RECORD        TYPE A2 IS RECORD
  B : INTEGER ;            B : FLOAT ;
END RECORD ;             END RECORD ;
C1 : A1 ;                C2 : A2 ;
```

Both fields B have different properties and therefore cannot use the same symbol table entry. The symbol table constructing routine allows identifiers with the same name at different lexical levels.

The field B is never referenced without some reference to the variable of type A1 or A2. The components of each record are linked to the record definition (see section 5.1 for structure). Therefore, for the duration of the type declaration, the lexical level is increased to store the components of the record. Once the record has been stored, the lexical level is decreased and all links from the hash table to the record components are removed. This results in the record components only being accessible through their type declarations. Once accessed, the component has all the properties of a normal variable.

The problem then became how to access the component again for use and type checking.

The only two ADA notations for using the components of a record are :

1) the point notation e.g. C1.B and

2) through the use of an aggregate e.g. C1(B=>2).

In both of these, the C1 is present and can be used to access B by tracing the pointers of the structure.

This is achieved by setting a flag when a '.' or '(' is encountered. Before processing the next token a check is performed to see if the token preceeding the '.' or '(' is a record, or is derived from a record. If it is, then the pointers are used to identify the next token and not the hash table method. This solution works well and allows different records to have fields with the same names and differing characteristics. In addition it allows the use of aggregates to set initial default values.

The other form of overloading that was overcome by this implementation was that of overloading within an arithmetic expression. This may occur across types where one operand is a numeric constant and the other is of a type derived from the same type as the numeric constant. This was achieved simply by evaluating both the type and base type of each operand. If one of the operands is a constant, then the base types are used for type compatibility, the result is a "constant", and has the type and base type of the original constant in the operation. This mechanism also caters for arithmetic where operands of different specified types are allowed and the result is also specified e.g. fixed * float = fixed.

EVALUATION OF NUMERIC LITERALS.

Evaluation of numeric literals is not done according to the ADA specification. The UCT-front end evaluates two types of numbers, integer and real. The integer numbers are referred to as ADA UNIVERSAL INTEGER and the real numbers as ADA UNIVERSAL FIXED. All numeric literals are constants. If evaluation of an

expression indicates that a particular constant should have had type float and not fixed, this is rectified.

Every entry in the symbol table is made up of the information relating to the entry followed by the ASCII representation of the token. The tokens are stored four characters to a word and from the token entry one knows how many words make up the token. Using this information each numeric string is converted into its equivalent decimal value. The method used is to decimalise each character by taking the numeric value of the character and multiplying this by the base of the number, creating a cumulative decimal equivalent of the input. e.g. The hexadecimal number 12 would be evaluated as 1*16+2 = decimal 18 and hexadecimal 122 would give ((1*16+2)*16)+2.

A check is made to ensure that no digit greater than the base of the number is used in its specification.

TYPE CHECKING.

Type checking covers many areas. There is arithmetic type checking, parameter type checking, subscript type checking, aggregate type checking, range checking, checking the type of initialising values and checking the type mark of a range constraint with the type of the range. This implementation does all of the above except the type checking for aggregates.

In the majority of instances the two operands must have the same type, but for derived types and to resove overloading, they must have the same base type. The major difficulty in implementing this occurs when the operands are array accesses. The array subscript type has to be matched with the array definition and the array component type with its value. The component type is only determined when the operator separating the operands is reached,

which may be after several other operators have been reached in the subscript.

Evaluating the type of an operand is a recursive procedure, and is repeated to find the type of each array subscript. Several levels of subscripting may exist and these levels are demarcated using the array of pointers.

All instances of type checking use the same procedure to find the type of the operand. The base type of the operand is also determined for use with overloading, derived and access types. Each subprogram declaration is linked in the same manner as a record declaration. Parameter checking therefore involves following the pointers of the structure to find the formal parameters, and matching these with the actual parameters when positional notation is used. Named notation is very similar except the structure may need to be traversed more than once if the named parameters are not in the same order as the formal parameters. The type compatibility between the parameter name and its value is checked, and this indirectly checks that the name is a valid formal parameter of the called procedure. The same method will be used for aggregates.

The ADA language specification explicitly states the typing rules for the arithmetic operators. These were implemented through a case statement, one case for each operator. This catered for all expressions where the type of the operands was universal integer or real, or both operands were of the identical type but not for expressions where the operand types were derived from the universal types. The base type of each operand was introduced to cater for instances of "legal" mixed mode arithmetic.

```
TYPE A IS 1 .. 10;
B : CONSTANT := 3;
C : INTEGER ;
D : A ;
BEGIN
  C := 5 ;  --LEGAL
  D := 3 ;  --LEGAL
  D := B ;  --LEGAL
  D := C ;  --ILLEGAL
END ;
```

Figure 5-3. EXAMPLE SHOWING ASSIGNMENT TYPE COMPATIBILITY.

Multiplication and division operators proved interesting. Operations using these operators are allowed between operands of the same base type. Mixed mode arithmetic with these is only allowed between integer and fixed point operands. If the operation between these two types is multiplication, the result is of type fixed. For division, only the left operand type may be fixed and the result is fixed. If both operands are fixed, the result is that of UNIVERSAL FIXED.

These rules have been implemented by flagging operands of type universal as constants, and using this information when storing the eventual type of the result.

Assignment is a special operation included in the case statement for operators. The same rules are applied when checking the type validity of variable initialisation. Two routines are required for these two occurrences of an assignment statement because the routine that determines the type of an operand uses the token stack, and relies on different kinds of tokens to mark the beginning of an operand. These markers are particular to the context.

A + B	OPERANDS ARE A	B
A(1)(B(1)).C-3	OPERANDS ARE A(1)(B(1)).C	3
A : B := NEW C	OPERANDS ARE B	C

Figure 5-4. EXAMPLE OF POSSIBLE OPERANDS.

Range checking requires some type checking and bound checking if the bounds are static expressions. The type checking is of two kinds:

1) The type mark of the range constraint must be compatible with the expressions denoting the range.

2) The expressions denoting the range must be type compatible.

The type checking routines use the same routine to determine the type of the operands. If none of the type checking fails, range checking is performed and the relevant error messages returned. An important area in this type checking is when to use the base type of the type mark. The bounds of a range are often of type UNIVERSAL INTEGER, FIXED or FLOAT and in these instances the base type must be used. This is not correct when variables of other types are used to define a range.

SUNDRY FUNCTIONS OF THE TOKEN MODULE.

Sundry functions of the TOKEN MODULE include: procedures giving order to enumeration types; checking valid use of procedure and entry names; storing the bounds for arrays and static expressions; evaluating expressions for storage in DIANA nodes; finding the type of number declarations; counting array bounds; evaluating and storing the expression after the DIGITS or DELTA in accuracy constraints; assigning the type and range of loop parameters; keeping note of whether an array is dynamic and has enough bounds specified for it.

6. CHAPTER 6. THE DIANA MODULE.

DIANA is the intermediate code used to create the interface of the UCT-ADA front end with the rest of the ADA compiler or ADA tools that may be designed. The code produced is in the form of an abstract syntax tree where each of the nodes of the tree have predefined attributes indicating certain semantic or syntactic properties.

This tree is represented by a linked list where each element of the list is a node in the tree. Each node has the same basic structure. The attributes of the nodes have various meanings depending on their node type.

APPENDIX IIa and IIb illustrate an example of some ADA fragments and their equivalent DIANA abstract syntax tree.

6.1. THE UCT IMPLEMENTATION OF DIANA.

Each node is represented by a record in PLUS (the language used for this module) and uses fifteen words of storage. The record is subdivided in the following manner:

1) Box-type. This is a nine bit integer used by the implementor to group alternate nodes with the same function in a structure. It is of no use to anyone else.

2) Box-kind. This nine bit integer is the unique number that identifies each node as a specific node type. Interpretation of this is vital to any user of the front end.

3) Reference. There are twelve of these eighteen bit integer fields present in each record. They represent pointers to other nodes in the tree. The first has the particular function of giving the node its unique sequence number for reference by other

nodes.

4) Value. Four one word value fields are provided to store any numeric values. They are defined to be of type real and any user of the DIANA tree can determine if the value stored was an integer by looking at the other attributes of the node.

5) Boolean. This is a one bit integer field that occurs six times. The first four occurences are linked to the value field. Every time a value is stored the corresponding boolean is set. This is necessary as the value field is initialised to zero. If the boolean is set the value was actually stored, even if it is zero. The remaining bits are for attributes that are stored as true or false.

6) Rep. Four one word logical fields store the representation of each token in the form it is input. If the token contains more than 16 characters, nodes will be linked together to store the remaining characters.

6.1.1. INTERPRETATION OF EACH NODE.

The box-type is of no use to anyone but the implementor so may be ignored. Appendix III gives a list of nodes (according to the DIANA MANUAL [Goos 1981]) with their corresponding unique box-kind number, and the structure of each node showing what references map to which attributes of each DIANA node.

Certain attributes have been catered for but not implemented yet so will not be mentioned. They are the attributes that contain certain semantic information, information that is machine dependent (i.e. cd-attributes in the manual) and any attributes that require a boolean result. The comment attribute and source position attribute have not been implemented but have been

provided for. The comment attribute is not mentioned whereas the source position is in APPENDIX III.

The references with particular meanings are:

One: This is the sequence number of the node and is generated by the program. Each new node gets a unique sequence number. These start from eleven and sequence numbers one to ten are used for implementation characteristics e.g. predefined type integer, float, string etc.

Eleven: If the token being stored has more than sixteen characters, special nodes are created to hold the subsequent groups of sixteen characters. These are linked through reference eleven of successive nodes. The number of nodes created will be sufficient to store the complete token.

Twelve: In ADA there may be lists of things. e.g. IDENTIFIER_LIST. These correspond to lists in DIANA. Each list has a particular list header. The header node links to the first element of the list through reference two, and this element links to the next through reference twelve. All remaining elements of the list are linked through reference twelve.

APPENDIX IIc illustrates the DIANA representation produced by the UCT-front end for the example given in APPENDIX IIa and IIb. DIANA nodes are produced as the ADA syntax demands. The calls to the DIANA producing module are interspersed in the parser and static semantic checking routines. Once a syntax or semantic error is found, the production of DIANA stops.

The DIANA MODULE has an array of size fifty. (This is set by a constant 'NO-OF-BOXES'). As each node is produced, it is stored in this array and kept there until it has been completed. Completed means that all possible attributes (structural and semantic) have been assigned a value, and any other node that

should be linked to it, either has been linked to it or its sequence number has been stored somewhere for reference. Once a node is complete it may be written out. Nodes are generally written out a structure at a time. E.g. The simple assignment statement

A := 3;

may generate the following nodes:

1) ASSIGN node.

2) USED-NAME-NODE node.

3) NUMERIC-LITERAL node.

Nodes two and three above may have been complex structures. E.g.

A.B := 3*4+5 ;

If this had been the statement, it would have been pointless to write out the box for A and 3, before completing the box for B and *4+5. The ASSIGN node will not be completed until the next statement is reached because it must be linked to the next statement. Writing out a "strucutre" means that when the next statement (or termination point) is reached, all nodes making up the previous statement or strucutre are written out.

Once a node has been written out, the array element that held that node is cleared to be re-used. The array thus functions as a circularly linked list.

Appendix IIc illustrates that the order in which the nodes of the DIANA tree are generated is haphazard when compared with the logical position of the nodes in the tree. This often implies that a tree is produced without there being a need for it. No other nodes reference it, and it just floats.

For example, as each part of an expression is parsed, a tree is constructed and the sequence number of the root is stored. A new root for the whole expression is created as the tree grows. This

new root must replace the old root. Once the complete expression has been parsed, that root must be linked as the beginning of the expression.

This is accomplished through a stack which keeps track of the root of the tree being dealt with. As a new root is formed, all references to the old root must be changed to point to the new root. All references to the expression must be to the root of the whole tree, which may only be known once the tree is completed.

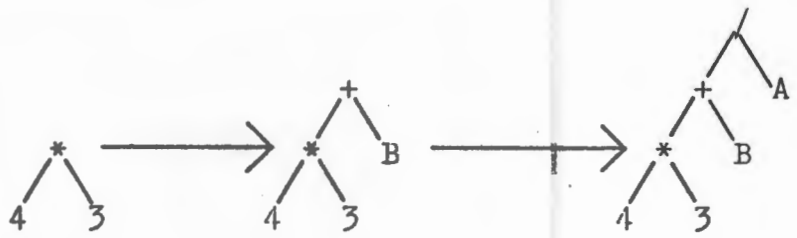


Figure 6-1. (4*3+B)/A

6.2. STACKS AND STRUCTURES USED IN THE DIANA MODULE.

One of the design principles of DIANA is "regularity" so structures produced are often very similar. ADA allows nesting of structures within each other. This makes scope a very important factor in the production of DIANA. Several different stacks have been used to overcome the problem of scope.

Stacks play a vital role in the DIANA module. They are used to prevent linking of structures across nesting levels, block levels and to aid linkage when complex structures are dealt with.

Every list in DIANA has a list header. Once the header has been linked to the first item of its list, there is no need to keep the header node in memory, but the position of the header node is required so that the entire list may be written out once the structure is complete. There is a stack which keeps the box-kind

of the headers that have been written out and their sequence numbers. Once their corresponding structures have been written out, they are removed from the stack.

Blocks may be nested within one another. The attributes and structures of the outermost block will be completed only when the inner blocks are complete. Linkage of nodes is not always in the backward direction, i.e. some attributes cannot be completed until further nodes have been generated. The number of these further nodes may be random, resulting in the necessity to search through the as yet incomplete nodes, find the required node and link it to the node just generated. There may be several incomplete nodes of the same kind, each belonging to a different block in the list. As the list is circular and nodes are not written out until complete, there is no guarantee that the first node found is the correct one.

The nesting of blocks is proper nesting. This was used to create a block-stack which holds the sequence number of the first node generated in each block. Every time a node is searched for (once the correct node kind is found) a check is done to ensure that the sequence number of the node is greater than or equal to the first sequence number of the block. Once a block is exited, all remaining nodes of that block are written out using the sequence number stored in the block-stack for that block. The stack top is then moved down by one.

The same idea is used with a statement stack. Each sequence of statements is linked together, and these may also be nested. Consider the following example. The line numbers are for reference.

```
     BEGIN
1    C := 0 ;
2    B := 0 ;
3    A := 3 ;
4    WHILE A < 10 LOOP
5        A := A + 1;
6        IF (A * 2) > 10 THEN
7           B := B + 1;
8           C := C + 2;
9        ELSE
10          B := B + 2;
11          C := C + 1;
12       END IF;
13   END LOOP ;
14   IF C > B THEN
15      A := C;
16   ELSE
17      A := B ;
18   END IF;
19   D := 10;
20   END ;
```

Figure 6-2.

In this example there are several sequences of statements. The following line numbers form the different sequences.

1) 1, 2, 3, 4, 14, 19
2) 5, 6
3) 7, 8
4) 10, 11
5) 15
6) 17

The most complex stack used is the one called "seq-no-store". ADA as a language is ambiguous. Many of the ambiguities were overcome by restructuring the syntax, or by using look-ahead. Consequently, in many productions a node is generated before the actual structure being parsed has been defined. The structure header is therefore only generated in the middle of the structure and must still be linked to its components, and other nodes must be linked to it. The production of DIANA often assumes that the next node to be generated will be the header node of the following structure.

Wherever this problem occurs in the ADA syntax, use is made of the "seq-no-store" stack. Productions are nested so a stack is ideal. Each time a production is entered and a node is generated before its header, the sequence number of this node is stored on the stack, and the stack top incremented. When the header is finally generated, the sequence number on the stack indicates the first component of that structure, and it also indicates the sequence number which has been referenced instead of the header. This means that all references to this sequence number can be changed to refer to the current header. All references on the stack are also changed so that if a new header should occur higher in the nesting structure, the stack will have a record of the existing header. This mechanism is used in the building of expression trees and name trees.

In expression, at each level of the parse tree, the stack is incremented and the sequence number of the next node to be generated is stored. As the parse tree backtracks, the stack is decremented level by level. The same is done with name.

In the following example of name, the kind of name is only determined at the '(' or '.'. E.g. A(F.W(I))

The DIANA nodes are generated in the following way:

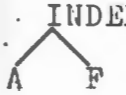
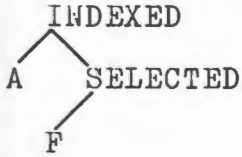
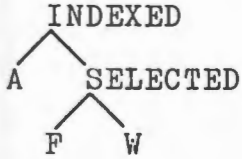
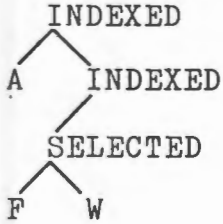
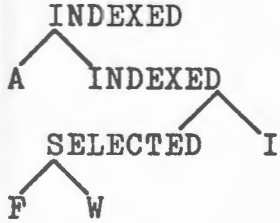
SEQUENCE NOS. ON STACK	ADA TOKEN	NODE SEQ-NO	TYPE OF NODE	TREE DEVELOPMENT
1	A	1	USED-NAME-ID	A
1	(	2	INDEXED	
1 3	F	3	USED-NAME-ID	
1 4	.	4	SELECTED	
1 4	W	5	USED-NAME-ID	
1 6	(	6	INDEXED	
1 6	I	7	USED-NAME-ID	
1	)			
	)			

Figure 6-3.

Stacks are therefore as important to the DIANA module as the token module is to the front end. All the problems created by the scope rules, nesting, and by ambiguities, have been solved using stacks.

6.3. STRUCTURE OF THE MODULE.

The DIANA module is made up of two major procedures with several others supporting them. Each procedure is one large case statement, where each case may be called from outside the module or within the module.

Procedure "genrate-diana-boxes" is the case statement which produces the nodes. It also calls the routines which manipulate the circularly linked list to determine where to put the newly created node.

Each unique box-kind is not produced by one particular constituent case. Several cases may produce the same box-kind but they may have different box-types, or acquire their attributes at different times, or be a particular box-kind that is produced in two different contexts.

The main function of this routine is to produce and complete the DIANA nodes concurrently as far as possible. Certain attributes can only be completed once several other nodes have been produced. Therefore there must be a further procedure that enables this procedure to search backwards through the nodes, find a particular node and complete the required attribute. The procedure that does this searches for a particular box-kind and then checks to see that the node found is in the correct block.

The second major procedure is called procdeure "polishing". Its purpose is to do any "tidying up" that is needed. The cases making up this case statement are called from particular places where special processing is required. When structures need to be written out, a call will be made to this routine which will find the required header node, see what its sequence number is and call the writing out routine. Certain attributes can be completed at

various points in the ADA syntax. At these points, this routine will be called and the relevant information completed.
All the stack incrementing and decrementing and setting and resetting of flags is done through this case statement. Semantic values are received and transferred to the relevant nodes through procedure "polishing".
If any attributes require alteration, this routine finds the relevant nodes and alters the fields. This is mainly used where the ADA structure cannot be identified immediately, and a basic structure is created. Once the actual structure has been identified, the basic structure is changed.
Consider the ADA syntax for the IDENTIFIER_LIST of a NUMBER_DECLARATION, OBJECT_DECLARATION, EXCEPTION_DECLARATION, COMPONENT_LIST and DISCRIMINANT_LIST. It is identical in all of the above examples and consequently they all generate the IDENTIFIER_LIST structure. DIANA differentiates between them all by generating NUMBER-ID, VAR-ID, EXCEPTION-ID, COMPONENT-ID and DISCRIMINANT-ID nodes. All these nodes have the same basic structure but are different box-kinds. At the end of each list, when it can be determined in the ADA syntax with which list is being dealt, procedure "polishing" is called and the relevant case finds the list header and changes all components of that list to their correct unique box-kind.

6.4. SEMANTIC INFORMATION.

An important feature of the DIANA tree is the semantic information it holds. The UCT-front end does the static semantic checking using the ADA symbol table and the results are passed to the DIANA structure (through procedure "polishing"). As this information

comes from the ADA symbol table, the symbol table must be visible at all times.

The UCT-front end uses the symbol table to store certain DIANA attributes that may be referenced by other nodes further down the tree once the original nodes have been output.

The attributes held in the symbol table used by the DIANA module are sequence numbers which point to various characteristics of a definition. They correspond to the following semantic attributes:

1) SM-DEFN. The sequence number where a particular identifier is declared.

2) SM-TYPE-STRUCTURE. Type structure refers to the node giving the structure of the type e.g. record, integer, array. Each time a new type is defined, sm-type-struct is set. For derived types or access types, it is the structure of the first type definition.

3) SM-BASE-TYPE. Contrary to type structure, this attribute is that of the last type declaration in a sequence of declarations.

4) SM-CONSTRAINT. This attribute is that of the most recent constraint imposed on the particular identifier.

The base type of the range node of a range or range constraint is also obtained from information held in the ADA symbol table.

6.5. OUTPUT OF THE DIANA TREE.

Each structure is written out as it is completed. This output is to a sequential disc file which must then be sorted with the key being each node sequence number for further use, or be used as a random access file. Nodes belonging to a particular structure will not necessarily be together on the file. In the formation of a list, as the nodes are linked, they are written out to avoid overwriting a link that already exists. Due to nesting, although

the body of a block may be written out, the header would only be written out much later when the next block at the same level was encountered.

Each record is output to a file as a string of characters. The record length is 120 characters. As PLUS has no random access facilities, a sequential file was produced.

The internal representation for DIANA used by the UCT-front end is in accordance with the specifications of the DIANA manual [Goos 1981]. The representation used for the DIANA nodes is similar to the structure of the nodes as specified in the manual and it would therefore be possible to write a driver routine which could translate the representation used into any desired form.

7. CHAPTER 7. WORKED EXAMPLES USING THE UCT-ADA FRONT END.

7.1. EXAMPLE 1.

This simple example illustrates the three areas covered by the UCT-front end.

Figure 7-1 is the program run with syntax errors. The error message indicates what the syntax error is and where error recovery begins.

Figure 7-2 is the program run with deliberate semantic errors. The first error is the simplest form of type incompatibility. The second is also a type mismatch but is not an obvious error. I-1.0 is legal, float - float. The error occurs because the function FACT returns an integer and therefore I*FACT( ) is illegal i.e. float * integer.

Figure 7-3 is the correct program compiled producing the DIANA given in figure 7-4.

Figure 7-1.

```
@RELOCATES.FINAL1,SL
ADA SCANNER VERSION U.C.T 1983 03/20/83  12:46:09
     1   1  0   --TEST2
     2   1  0   FUNCTION FACT(I: INTEGER) RETURN INTEGER IS
     3   2  0   BEGIN
     4   2  0     IF I = 1
     5   2  0       RETURN 1 ;
***ERROR*** 182 'THEN' EXPECTED , RESERVED-WORD RETURN AT
LINE 5 COL 5 SKIPS AND RECOVERS TO RETURN AT LINE 5 COL 5
     6   2  0     ELSE
     7   2  0       RETURN I*FACT(I-1) ;
     8   2  0     END IF
     9   2  0   END FACT ;
***ERROR*** 202 ';' EXPECTED , RESERVED-WORD END AT
LINE 9 COL 1 SKIPS AND RECOVERS TO END AT LINE 9 COL 1
***** END PROCESSING 2 ERRORS
```

Figure 7-2.

```
@RELOCATES.FINAL1,SL
ADA SCANNER VERSION U.C.T 1983 03/20/83  12:49:37
     1   1  0   --TEST2
     2   1  0   FUNCTION FACT(I: FLOAT) RETURN INTEGER IS
     3   2  0   BEGIN
     4   2  0     IF I = 1 THEN
***ERROR*** 504 OPERANDS OF DIFFERENT TYPES.
     5   2  0       RETURN 1 ;
     6   2  0     ELSE
     7   2  0       RETURN I*FACT(I-1.0) ;
***ERROR*** 510 OPERANDS OF ILLEGAL TYPES FOR THIS OPERATION .
     8   2  0     END IF ;
     9   2  0   END FACT ;
***** END PROCESSING 2 ERRORS
```

Figure 7-3.

```
@RELOCATES.FINAL1,SL
ADA SCANNER VERSION U.C.T 1983 03/20/83  12:39:49
     1   1  0   --TEST2
     2   1  0   FUNCTION FACT(I: INTEGER) RETURN INTEGER IS
     3   2  0   BEGIN
     4   2  0     IF I = 1 THEN
     5   2  0       RETURN 1 ;
     6   2  0     ELSE
     7   2  0       RETURN I*FACT(I-1) ;
     8   2  0     END IF ;
     9   2  0   END FACT ;
***** END PROCESSING 0 ERRORS
```

Figure 7-4.

```
    28    11    12     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
   110    15    16     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
    72    16    17    19    21     0     0     0     0     0     0
0     0.00000     .00000    00
    70    17    18     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
    73    18    19    21     0     0     0     0     0     0     0
0     0.00000     .00000    00I
    33    19    20     0     4     4     0     0     0     0     0
0     0.00000     .00000    00
   157    20     4     0     0     0     0     0     0     0     0
0     0.00000     .00000    00INTEGER
   166    21     0     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
    61    13    15    22     0     0     0     0     0     0     0
0     0.00000     .00000    00
    63    14    13    25     0     0     0     0     0     0     0
0     0.00000     .00000    00FACT
    33    22    23     0     4     4     0     0     0     0     0
0     0.00000     .00000    00
   157    23     4     0     0     0     0     0     0     0     0
0     0.00000     .00000    00INTEGER
   137    24    14    13    25     0     0     0     0     0     0
0     0.00000     .00000    00
   134    26    27     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
   134    34    35     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
   126    35    36     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
   100    36     4     0     0     0     0     0     0     0     0
0     0.10000+001.00000    101
    29    28    30    34     0     0     0     0     0     0     0
0    37.00000     .00000    00
```

```
    29    37    38    39     0     0     0     0     0     0     0
0     0.00000     .00000    00
   134    39    40     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
   109    47    49     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
   126    40    42     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
   157    41    18     0     0     0     0     0     0     0     0
0    45.00000     .00000    00I
    62    42    43    44     0     0     0     0     0     0     0
0     0.00000     .00000    00
   138    43     0     0     0     0     0     0     0     0     0
0     0.00000     .00000    00*
   109    44    41     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
    62    45    46    47     0     4     0     0     0     0     0
0     0.00000     .00000    00
   157    46    14     0     0     0     0     0     0     0     0
0     0.00000     .00000    00FACT
   157    48    18     0     0     0     0     0     0     0     0
0    52.00000     .00000    00I
    62    49    50    51     0     0     0     0     0     0     0
0     0.00000     .00000    00
   138    50     0     0     0     0     0     0     0     0     0
0     0.00000     .00000    00-
   109    51    48     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
   100    52     4     0     0     0     0     0     0     0     0
0     0.10000+001.00000    101
    71    27    28     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
   157    29    18     0     0     0     0     0     0     0     0
0    33.00000     .00000    00I
    62    30    31    32     0     0     0     0     0     0     0
0     0.00000     .00000    00
   138    31     0     0     0     0     0     0     0     0     0
0     0.00000     .00000    00=
   109    32    29     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
   100    33     4     0     0     0     0     0     0     0     0
0     0.10000+001.00000    101
   166    38     0     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
    19    25     0    26     0     0     0     0     0     0     0
0     0.00000     .00000    00
    27    12     0     0    24     0     0     0     0     0     0
0     0.00000     .00000    00
```

7.2. EXAMPLE 2.

This example introduces interesting features of ADA, that of an incomplete type declaration, and the access type declaration. Default initial values are specified for the record LEADER and a function is defined and called within the program.

Figure 7-5 illustrates syntactic and semantic errors. Notice that because of the error in the definition of TREF, all variables of type TREF are not recognised as record variables. As a result of the method used to store records, the record components can only be accessed through the record variable name resulting in the generation of several unneccessary error messages.

Figure 7-6 is the syntactically correct program and APPENDIX IVa is the DIANA output by the UCT-front end.

Figure 7-5.

```
@RELOCATES.FINAL1,SL
ADA SCANNER VERSION U.C.T 1983 03/21/83  22:12:43
     1   1  0   --EX91--
     2   1  0   PROCEDURE TSORT IS --TOPOLOGICAL SORT
     3   1  0     TYPE LEADER ;
     4   2  0     TYPE TRAILER ;
     5   2  0     TYPE LREF IS ACCESS LEADER
     6   2  0     TYPE TREF IS ACCESS  ;
***ERROR*** 202 ';' EXPECTED , RESERVED-WORD TYPE AT
LINE 6 COL 3 SKIPS AND RECOVERS TO TYPE AT LINE 6 COL 3
***ERROR*** 260 <TYPE MARK> EXPECTED , SPECIAL ; AT
LINE 6 COL 24 SKIPS AND RECOVERS TO ; AT LINE 6 COL 24
     7   2  0     TYPE LEADER IS
     8   2  0      RECORD
     9   3  0        KEY      : INTEGER   := 0;
    10   3  0        COUNT    : INTEGER   := 0 ;
    11   3  0        TRAIL    : TREF      := NULL;
    12   3  0        NXT     : LREF      := NULL;
    13   3  0      END RECORD;
    14   2  0     TYPE TRAILER IS
    15   2  0      RECORD
    16   4  0        ID       : LREF ;
    17   4  0        NXT     : TREF ;
    18   4  0      END RECORD ;
    19   2  0     HEAD,TAIL,P,Q : LREF;
    20   2  0     T             : TREF;
    21   2  0     Z             : INTEGER ;
    22   2  0     X,Y           : INTEGER;
    23   2  0   FUNCTION L(W:INTEGER) RETURN LREF IS
    24   5  0     H : LREF ;
    25   5  0     BEGIN
    26   5  0       H := HEAD ;
    27   5  0       TAIL.KEY := H;
***ERROR*** 508 ASSIGNMENT ATTEMPTED BETWEEN OPERANDS OF DIFFERENT
TYPES .
    28   5  0       WHILE TAIL.KEY /= T LOOP
***ERROR*** 504 OPERANDS OF DIFFERENT TYPES.
    29   6  0         H := H.NXT;
    30   6  0       END LOOP;
    31   5  0       IF H = TAIL
    32   5  0          TAIL := NEW LEADER;
***ERROR*** 182 'THEN' EXPECTED , VARIABLE-ID TAIL AT
LINE 32 COL 8 SKIPS AND RECOVERS TO TAIL AT LINE 32 COL 8
    33   5  0          Z := Z + 1;
    34   5  0          H.COUNT := 0;
    35   5  0          H.TRAIL := NULL ;
    36   5  0          H.NXT := TAIL;
    37   5  0       END IF ;
    38   5  0     RETURN H;
    39   5  0     END L;
    40   2  0   BEGIN
    41   2  0     HEAD := NEW LEADER;
    42   2  0     TAIL := HEAD;
    43   2  0     Z := 0;
    44   2  0     GET(X);
    45   2  0     WHILE X /= 0 LOOP
```

```
    46   7  0          GET(Y) ;
    47   7  0          PUT(X); PUT(Y);
    48   7  0          P := L(1.1);
***ERROR*** 534 FORMAL AND ACTUAL PARAMETERS MIS-MATCHED
    49   7  0          Q := L(Y);
    50   7  0          T := NEW TRAILER(ID => NULL, NXT = NULL);
***ERROR*** 510 OPERANDS OF ILLEGAL TYPES FOR THIS OPERATION .
    51   7  0          T.ID := Q;
***ERROR*** 156 UNRECOGNIZABLE STATEMENT , IDENTIFIER ID AT
LINE 51 COL 7 SKIPS AND RECOVERS TO ID AT LINE 51 COL 7
***ERROR*** 528 UNDECLARED IDENTIFIER .
    52   7  0          T.NXT := P.TRAIL;
***ERROR*** 156 UNRECOGNIZABLE STATEMENT , IDENTIFIER NXT AT
LINE 52 COL 7 SKIPS AND RECOVERS TO NXT AT LINE 52 COL 7
***ERROR*** 528 UNDECLARED IDENTIFIER .
    53   7  0          P.TRAIL := T;
    54   7  0          Q.COUNT := Q.COUNT +1;
    55   7  0          GET(X);
    56   7  0        END LOOP;
    57   2  0       P := HEAD;
    58   2  0       HEAD := NULL;
    59   2  0        WHILE P /= TAIL LOOP
    60   8  0          Q := P ;
    61   8  0          P := P.NXT;
    62   8  0          IF Q.COUNT = 0 THEN
    63   8  0           Q.NXT := HEAD;
    64   8  0           HEAD := Q;
    65   8  0          END IF ;
    66   8  0        END LOOP ;
    67   2  0        Q := HEAD ;
    68   2  0      OUTER : WHILE Q /= NULL LOOP
    69   9  0                  PUT(Q.KEY);
    70   9  0                  Z := Z - 1;
    71   9  0                  T := Q.TRAIL;
    72   9  0                  Q := Q.NXT;
    73   9  0      INNER :     WHILE T /= NULL LOOP
    74  10  0                    P := T.ID;
***ERROR*** 156 UNRECOGNIZABLE STATEMENT , IDENTIFIER ID AT
LINE 74 COL 21 SKIPS AND RECOVERS TO ID AT LINE 74 COL 21
***ERROR*** 532 INVALID ENTRY CALL OR PROCEDURE CALL.
    75  10  0                    P.COUNT := P.COUNT -1;
    76  10  0                    IF P.COUNT = 0 THEN
    77  10  0                       P.NXT := Q ;
    78  10  0                       Q := P;
    79  10  0                    END IF;
    80  10  0                    T := T.NXT;
***ERROR*** 156 UNRECOGNIZABLE STATEMENT , IDENTIFIER NXT AT
LINE 80 COL 21 SKIPS AND RECOVERS TO NXT AT LINE 80 COL 21
***ERROR*** 532 INVALID ENTRY CALL OR PROCEDURE CALL.
    81  10  0                  END LOOP  INNER;
    82   9  0                END LOOP OUTER;
    83   2  0       IF Z = 0 THEN
    84   2  0       PUT("THIS SET IS NOT PARTIALLY ORDERED.");
    85   2  0       END IF ;
    86   2  0      END TSORT;
***** END PROCESSING 15 ERRORS
```

Figure 7-6.

```
@RELOCATES.FINAL1,SL
ADA SCANNER VERSION U.C.T 1983 03/21/83  22:24:16
     1   1  0   --EX91--
     2   1  0   PROCEDURE TSORT IS --TOPOLOGICAL SORT
     3   1  0     TYPE LEADER ;
     4   2  0     TYPE TRAILER ;
     5   2  0     TYPE LREF IS ACCESS LEADER ;
     6   2  0     TYPE TREF IS ACCESS TRAILER ;
     7   2  0     TYPE LEADER IS
     8   2  0      RECORD
     9   3  0        KEY       : INTEGER    := 0;
    10   3  0        COUNT     : INTEGER    := 0 ;
    11   3  0        TRAIL     : TREF       := NULL;
    12   3  0        NXT       : LREF       := NULL;
    13   3  0      END RECORD;
    14   2  0     TYPE TRAILER IS
    15   2  0      RECORD
    16   4  0        ID        : LREF ;
    17   4  0        NXT       : TREF ;
    18   4  0      END RECORD ;
    19   2  0     HEAD,TAIL,P,Q : LREF;
    20   2  0     T             : TREF;
    21   2  0     Z             : INTEGER ;
    22   2  0     X,Y           : INTEGER;
    23   2  0   FUNCTION L(W:INTEGER) RETURN LREF IS
    24   5  0     H : LREF ;
    25   5  0     BEGIN
    26   5  0       H := HEAD ;
    27   5  0       TAIL.KEY := W;
    28   5  0       WHILE TAIL.KEY /= W LOOP
    29   6  0         H := H.NXT;
    30   6  0       END LOOP;
    31   5  0       IF H = TAIL
    32   5  0       THEN
    33   5  0          TAIL := NEW LEADER;
    34   5  0          Z := Z + 1;
    35   5  0          H.COUNT := 0;
    36   5  0          H.TRAIL := NULL ;
    37   5  0          H.NXT := TAIL;
    38   5  0       END IF ;
    39   5  0     RETURN H;
    40   5  0     END L;
    41   2  0   BEGIN
    42   2  0     HEAD := NEW LEADER;
    43   2  0     TAIL := HEAD;
    44   2  0     Z := 0;
    45   2  0     GET(X);
    46   2  0     WHILE X /= 0 LOOP
    47   7  0       GET(Y) ;
    48   7  0       PUT(X); PUT(Y);
    49   7  0       P := L(X);
    50   7  0       Q := L(Y);
    51   7  0       T := NEW TRAILER(ID => NULL, NXT = NULL);
    52   7  0       T.ID := Q;
    53   7  0       T.NXT := P.TRAIL;
    54   7  0       P.TRAIL := T;
```

```
    55    7  0        Q.COUNT := Q.COUNT +1;
    56    7  0        GET(X);
    57    7  0      END LOOP;
    58    2  0     P := HEAD;
    59    2  0     HEAD := NULL;
    60    2  0      WHILE P /= TAIL LOOP
    61    8  0        Q := P ;
    62    8  0        P := P.NXT;
    63    8  0        IF Q.COUNT = 0 THEN
    64    8  0         Q.NXT := HEAD;
    65    8  0         HEAD := Q;
    66    8  0        END IF ;
    67    8  0      END LOOP ;
    68    2  0      Q := HEAD ;
    69    2  0    OUTER : WHILE Q /= NULL LOOP
    70    9  0                PUT(Q.KEY);
    71    9  0                Z := Z - 1;
    72    9  0                T := Q.TRAIL;
    73    9  0                Q := Q.NXT;
    74    9  0    INNER :     WHILE T /= NULL LOOP
    75   10  0                  P := T.ID;
    76   10  0                  P.COUNT := P.COUNT -1;
    77   10  0                  IF P.COUNT = 0 THEN
    78   10  0                     P.NXT := Q ;
    79   10  0                     Q := P;
    80   10  0                  END IF;
    81   10  0                  T := T.NXT;
    82   10  0                END LOOP  INNER;
    83    9  0              END LOOP OUTER;
    84    2  0     IF Z = 0 THEN
    85    2  0     PUT("THIS SET IS NOT PARTIALLY ORDERED.");
    86    2  0     END IF ;
    87    2  0    END TSORT;
***** END PROCESSING 0 ERRORS
```

7.3. EXAMPLE 3.

This example illustrates that the syntax checker validates the syntax for most of ADA and not just the "PASCAL" subset. An interesting feature of this program is the use of an aggregate to initialise the values of the array of type WORK-WEEK.

Figure 7-7.

```
@RELOCATES.FINAL1,SL
ADA SCANNER VERSION U.C.T 1983 03/21/83  22:44:57
     1   1  0   PACKAGE WORK_DATA IS
     2   1  0     TYPE DAY IS(MON,TUES,WED,THU,FRI,SAT,SUN);
     3   1  0     TYPE TIME IS DELTA 0.01 RANGE 0.0 .. 24.0 ;
     4   1  0     TYPE WORK_WEEK IS ARRAY(MON .. SUN) OF TIME ;
     5   1  0     NORMAL_HOURS : CONSTANT WORK_WEEK := (MON .. FRI
                              => 8.0 , SAT .. SUN => 0.0)
     6   1  0   END ;
     7   1  0   FUNCTION OVERTIME(ACTUAL: WORK_WEEK)
                              RETURN FLOAT IS
     8   2  0   OVER : FLOAT := 0.0;
     9   2  0   I : INTEGER ;
    10   2  0   BEGIN
    11   2  0     FOR I IN DAY LOOP
    12   3  0       OVER := OVER + FLOAT(ACTUAL(I))
                        - FLOAT(NORMAL_HOURS(I))
    13   3  0     END LOOP;
    14   2  0     RETURN OVER ;
    15   2  0   END OVERTIME ;
    16   1  0   FUNCTION SALARY(RATE: FLOAT ;
           HOURS : WORK_WEEK)RETURN FLOAT IS
    17   4  0   PAY, EXCESS : FLOAT ;
    18   4  0   BEGIN
    19   4  0     EXCESS := OVERTIME(HOURS);
    20   4  0     IF EXCESS < 0.0 THEN
    21   4  0       PAY := (40.0 + EXCESS) * RATE;
    22   4  0     ELSE
    23   4  0       PAY := 40.0*RATE + 2.0*EXCESS*RATE;
    24   4  0     END IF;
    25   4  0     RETURN PAY;
    26   4  0   END SALARY;
***** END PROCESSING 0 ERRORS
```

7.4. EXAMPLE 4.

This example again includes features not covered by the semantic and code generation phases of this thesis. It was used to check that loop parameters could be used in expressions with the loop parameter having the correct default type.

Figure 7-8.

```
@RELOCATES.FINAL1,SL
ADA SCANNER VERSION U.C.T 1983 03/21/83  22:58:19
     1   1  0   PACKAGE ARRAYTEST IS PRIVATE END ;
     2   1  0   PACKAGE BODY ARRAYTEST IS
     3   2  0   LOOPARRAY : ARRAY ( 1 .. 10 ) OF INTEGER  ;
     4   2  0   J : INTEGER ;
     5   2  0   BEGIN
     6   2  0   FOR I IN 1 .. 10 LOOP
     7   3  0   LOOPARRAY (I) := I ;
     8   3  0   END LOOP ;
     9   2  0   FOR J IN 3 .. 12 LOOP
    10   4  0   LOOPARRAY ( J - 3 ) := LOOPARRAY ( J -3)
                  + LOOPARRAY(J)
    11   4  0   END LOOP ;
    12   2  0   END ;
***** END PROCESSING 0 ERRORS
```

7.5. EXAMPLE 5.

Figure 7-9 illustrates other errors that may occur.

Figure 7-10 is the corrected program and the DIANA produced may be found in APPENDIX IVb.

Figure 7-9.

```
@RELOCATES.FINAL1,SL
ADA SCANNER VERSION U.C.T 1983 03/21/83  23:21:29
     1   1  0   --EX90--
     2   1  0   PROCEDURE INSERT IS --LINEAR LIST INSERTION.
     3   1  0    TYPE NODE;
     4   2  0    TYPE REF IS ACCESS NODE ;
     5   2  0    TYPE NODE IS
     6   2  0      RECORD
     7   3  0         KEY : INTEGER ;
     8   3  0         COUNT : INTEGER ;
     9   3  0         NXT : REF ;
    10   3  0     END RECORD ;
    11   2  0     K : INTEGER ;
    12   2  0     ROOT : REF := NEW NODE(0,0,NULL) ;
    13   2  0     PROCEDURE SEARCH(X : INTEGER ; BASE :
                   IN OUT REF) IS
    14   4  0       W : REF ;
    15   4  0       B : BOOLEAN ;
    16   4  0       BEGIN
    17   4  0         W := BASE ;
    18   4  0         B := TRUE ;
    19   4  0         WHILE W   LOOP
***ERROR*** 514 CONDITION MUST BE A BOOLEAN EXPRESSION.
    20   5  0           IF W.KEY = X
    21   5  0           THEN B := FALSE ;
    22   5  0           ELSE W := W.NXT ;
    23   5  0           END IF ;
    24   5  0         END LOOP ;
    25   4  0         IF B THEN -- NEW ENTRY
    26   4  0           W := BASE ;
    27   4  0           BASE := NEW NODE(0,0,NULL) ;
    28   4  0           BASE.KEY := X ;
    29   4  0           BASE.COUNT := 1.1 ;
***ERROR*** 508 ASSIGNMENT ATTEMPTED BETWEEN OPERANDS OF DIFFERENT
TYPES .
    30   4  0         ELSE
    31   4  0           W.COUNT := W.COUNT + 1;
    32   4  0         END IF ;
    33   4  0     END SEARCH ;
    34   2  0     PROCEDURE PRINTLIST(V:REF) IS
    35   6  0       BEGIN
    36   6  0       WHILE V /= NULL LOOP
    37   7  0         PUT(V.KEY);
    38   7  0         PUT(V.COUNT);
    39   7  0         V := V.NXT ;
    40   7  0       END LOOP ;
```

```
    41   6  0      END PRINTLIST;
    42   2  0    BEGIN -- MAIN PROGRAM
    43   2  0      GET(K);
    44   2  0      WHILE K /= 0
    45   8  0        SEARCH(K,ROOT);
***ERROR*** 150 'LOOP' EXPECTED , PROCEDURE-ID SEARCH AT
LINE 45 COL 5 SKIPS AND RECOVERS TO SEARCH AT LINE 45 COL 5
    46   8  0        GET(K) ;
    47   8  0      END LOOP ;
    48   2  0      PRINTLIST(ROOT) ;
    49   2  0    END INSERT ;
***** END PROCESSING 3 ERRORS
```

Figure 7-10.

```
@RELOCATES.FINAL1,SL
ADA SCANNER VERSION U.C.T 1983 03/21/83  23:04:33
     1    1  0   --EX90--
     2    1  0   PROCEDURE INSERT IS --LINEAR LIST INSERTION.
     3    1  0    TYPE NODE;
     4    2  0    TYPE REF IS ACCESS NODE ;
     5    2  0    TYPE NODE IS
     6    2  0      RECORD
     7    3  0         KEY : INTEGER ;
     8    3  0         COUNT : INTEGER ;
     9    3  0         NXT : REF ;
    10    3  0      END RECORD ;
    11    2  0      K : INTEGER ;
    12    2  0      ROOT : REF := NEW NODE(0,0,NULL) ;
    13    2  0      PROCEDURE SEARCH(X : INTEGER ; BASE :
                     IN OUT REF) IS
    14    4  0        W : REF ;
    15    4  0        B : BOOLEAN ;
    16    4  0        BEGIN
    17    4  0          W := BASE ;
    18    4  0          B := TRUE ;
    19    4  0          WHILE W /= NULL AND B LOOP
    20    5  0            IF W.KEY = X
    21    5  0            THEN B := FALSE ;
    22    5  0            ELSE W := W.NXT ;
    23    5  0            END IF ;
    24    5  0          END LOOP ;
    25    4  0          IF B THEN -- NEW ENTRY
    26    4  0            W := BASE ;
    27    4  0            BASE := NEW NODE(0,0,NULL) ;
    28    4  0            BASE.KEY := X ;
    29    4  0            BASE.COUNT := 1 ;
    30    4  0          ELSE
    31    4  0            W.COUNT := W.COUNT + 1;
    32    4  0          END IF ;
    33    4  0      END SEARCH ;
    34    2  0      PROCEDURE PRINTLIST(V:REF) IS
    35    6  0         BEGIN
    36    6  0         WHILE V /= NULL LOOP
    37    7  0           PUT(V.KEY);
    38    7  0           PUT(V.COUNT);
    39    7  0           V := V.NXT ;
    40    7  0         END LOOP ;
    41    6  0      END PRINTLIST;
    42    2  0    BEGIN -- MAIN PROGRAM
    43    2  0      GET(K);
    44    2  0      WHILE K /= 0 LOOP
    45    8  0        SEARCH(K,ROOT);
    46    8  0        GET(K) ;
    47    8  0      END LOOP ;
    48    2  0      PRINTLIST(ROOT) ;
    49    2  0    END INSERT ;
***** END PROCESSING 0 ERRORS
```

7.6. EXAMPLE 6.

EXAMPLE 6 is not a sensible example and was produced to illustrate the DIANA produced for sequences of statements. This may be found in APPENDIX IVc.

Figure 7-11.

```
@RELOCATES.FINAL1,SL
ADA SCANNER VERSION U.C.T 1983 03/21/83  23:26:19
     1   1  0   PROCEDURE V IS
     2   1  0   TYPE INTEGER IS RANGE 1 .. 50 ;
     3   2  0   TYPE COLOUR IS (RED, ORANGE, YELLOW,GREEN) ;
     4   2  0   ROBOT : COLOUR ;
     5   2  0   VAR2 , VAR1 : INTEGER ;
     6   2  0   BEGIN
     7   2  0   <<LAB1>> VAR1 := 2 ;
     8   2  0   <<LAB2>> <<LAB3>> VAR2 := VAR1 ;
     9   2  0   VAR2 := 10 ;
    10   2  0   NULL ;
    11   2  0   GOTO LAB2 ;
    12   2  0   GOTO LAB1 ;
    13   2  0   GOTO LAB3 ;
    14   2  0   <<LAB4>> NAMED1 : LOOP
    15   3  0              CASE VAR1 IS
    16   3  0              WHEN 1\2 =>
    17   3  0                        DECLARE
    18   4  0                        VAR4 : FLOAT ;
    19   4  0                      BEGIN
    20   4  0                          VAR4 := 4.5 * 3.3 ;
    21   4  0                          VAR4 := VAR4 ** 2 ;
    22   4  0                        END ;
    23   3  0              WHEN 3 => LOOP
    24   5  0                        VAR1 := VAR1 + 1 ;
    25   5  0                        EXIT WHEN VAR1 = 10 ;
    26   5  0                      END LOOP ;
    27   3  0              WHEN OTHERS => NULL ;
    28   3  0              END CASE ;
    29   3  0              VAR1 := VAR2 ;
    30   3  0              VAR2 := 20 ;
    31   3  0              EXIT NAMED1 ;
    32   3  0              END LOOP NAMED1 ;
    33   2  0   NAMED2 : FOR I IN COLOUR LOOP
    34   6  0              VAR1 := VAR1 + 1 ;
    35   6  0              VAR2 := VAR2 + VAR1 ;
    36   6  0              EXIT NAMED2 WHEN VAR1 = 3 ;
    37   6  0              END LOOP ;
    38   2  0   NAMED3 : DECLARE
    39   7  0              VAR3 : FLOAT ;
    40   7  0              BEGIN
    41   7  0                VAR3 := 3.3 ;
    42   7  0                IF VAR1 = VAR2
    43   7  0                THEN
    44   7  0                   VAR3 := 6.6 ;
    45   7  0                END IF ;
```

```
    46   7  0            END ;
    47   2  0   WHILE VAR1 < 10 LOOP
    48   8  0                    IF VAR2 = 10
    49   8  0                    THEN
    50   8  0                       VAR1 := 5 ;
    51   8  0                    ELSIF VAR2 = 9
    52   8  0                    THEN
    53   8  0                       VAR1 := 7 ;
    54   8  0                    ELSIF VAR2 = 7
    55   8  0                    THEN
    56   8  0                       VAR1 := 5 ;
    57   8  0                    ELSE
    58   8  0                       VAR1 := 3 ;
    59   8  0                    END IF ;
    60   8  0                    EXIT ;
    61   8  0                    END LOOP ;
    62   2  0   FOR J IN REVERSE GREEN .. RED LOOP
    63   9  0                                 ROBOT := J ;
    64   9  0                                 RETURN ;
    65   9  0                                 RETURN VAR1 ;
    66   9  0                                 NULL ;
    67   9  0                                 END LOOP ;
    68   2  0   ROBOT := GREEN ;
    69   2  0   END ;
***** END PROCESSING 0 ERRORS
```

8. CHAPTER 8. CONCLUSION.

The purpose of this thesis was the production of a front end processor for a subset of ADA. This goal has been achieved for the "PASCAL" subset of ADA with several other structures included but not explicitly mentioned. The processor runs on a Univac 1100 system and was produced by using a compiler generator package for the syntax checker, PLUS routines for the semantic analysis and code generation. It produces an internal form of DIANA consistent with the DIANA specification.

This output is being used by several honours students at the University of Cape Town who are designing programming environment tools for ADA.

Experience has been gained in using the compiler generator package GSA 1100, and many of the problems encountered by the implementors at the University of Cape Town have been subsequently reported in the literature [Bonet] [Harrison 1982] [Winkler 1981]. It appears that the solutions used by other implementors coincide favourably with those used in the UCT-front end processor.

In undertaking this project greater insight has been achieved into the ADA programming language, DIANA, and the area of program environments. The project provides a basis for further research.

9. APPENDIX I. THE LEXICAL SPECIFICATION FOR UCT-ADA.

```
$OPTS       LEX
            ASCII

$PARAM      END_FILE
            BASED_NUMBER
            CHARACTER_LITERAL
            CHARACTER_STRING
            DECIMAL_NUMBER
            IDENTIFIER
            ILLEGAL
            SPECIAL

$NON-TERM   <TOKEN>

$SET        $[ILLEGALS]$
            $[DIGIT]$
            $[LETTER]$
            $[SIGN]$
            $[SPECIALS]$
            $[STRINGCHARS]$
            $[COMMENTCHARS]$
            $[BASEARITH]$
            $[CHARACTERLIT]$

$PROD
<TOKEN> ::=  ^ TOKEN #  ( $[ SP EOL EOC ]$ -> TOKEN ) **

[$] [

! $[LETTER]$ ( '_' [ !  $[  $[LETTER]$ $[DIGIT]$ ]$ \
                                  <* 20 => ILLEGAL ]  \
                    ! $[  $[LETTER]$ $[DIGIT]$ ]$ ) ...
                                  => IDENTIFIER \

$[SPECIALS]$                      => SPECIAL    \
'=' ( '>' \ ':' )                 => SPECIAL    \
'.' ( '.' )                       => SPECIAL    \
'*' ( '*' )                       => SPECIAL    \
':' ( '=' )                       => SPECIAL    \
'/' ( '=' )                       => SPECIAL    \
'>' ( '=' \ '>' )                 => SPECIAL    \
'<' ( '=' \ '<' \ '>' )           => SPECIAL    \
'(' ( ')' )                       => SPECIAL    \
# '!' $ '¦'                       => SPECIAL    \

? EOF $ '@' $ 'E' $ 'O' $ 'F'     => END_FILE   \

$[ILLEGALS]$                <* 55 => ILLEGAL    \


''' ?[ $[CHARACTERLIT]$ ''' :
            $[CHARACTERLIT]$ ''' => CHARACTER_LITERAL ]?
                                 => SPECIAL \

'"' ^ LIT [ $[STRINGCHARS]$ -> LIT \ '"' \ ? EOL <* 10 $ '"' \
? $[ EOF .. BAD ]$ <* 40 $ '"' ] ( # '"' -> LIT )
```

```
                                        => CHARACTER_STRING \

$[DIGIT]$ ( # '_' [ $[DIGIT]$ \ <* 30 => ILLEGAL ] \ $[DIGIT]$ )
...
[ [ '#' \ # ':' $ '#' ] [ $[ $[DIGIT]$ $[BASEARITH]$ ]$
                          \ <* 35 => ILLEGAL ]
( # '_' [ $[ $[DIGIT]$ $[BASEARITH]$ ]$ \ <* 30 => ILLEGAL ] \
                                $[ $[DIGIT]$ $[BASEARITH]$ ]$ ) ...
?[ '.' '.' :  => BASED_NUMBER ]?
( '.' [ $[ $[DIGIT]$ $[BASEARITH]$ ]$ \ <* 35 => ILLEGAL ]
( # '_' [ $[ $[DIGIT]$ $[BASEARITH]$ ]$
\ <* 30 => ILLEGAL ] \ $[ $[DIGIT]$ $[BASEARITH]$ ]$ ) ... )
[ [ '#' \ # ':' $ '#' ] \ <* 36 => ILLEGAL ]
( ! $[ 'e' 'E' ]$ ( $[SIGN]$ ) [ $[DIGIT]$ \ <* 35 => ILLEGAL ]
( # '_' [ $[DIGIT]$ \ <* 30 => ILLEGAL ] \ $[DIGIT]$ ) ... )
                                        => BASED_NUMBER \
 ?[ '.' '.' :  => DECIMAL_NUMBER ]?
                ( '.' [ $[DIGIT]$ \ <* 35 => ILLEGAL ]
( # '_' [ $[DIGIT]$ \ <* 30 => ILLEGAL ] \ $[DIGIT]$ ) ... )
( ! $[ 'E' 'e' ]$ ( $[SIGN]$ ) [ $[DIGIT]$ \ <* 35 => ILLEGAL ]
( # '_' [ $[DIGIT]$ \ <* 30 => ILLEGAL ] \ $[DIGIT]$ ) ... ) ]
                                        => DECIMAL_NUMBER \

'-' [ '-' # ( $[COMMENTCHARS]$ ) ...
[ EOL -> TOKEN \ ** <* 50 -> TOKEN ] \ => SPECIAL ] ] .

$EJECT
$[SIGN]$          ::= $[ '+' '-' ]$ .

$[LETTER]$        ::= $[ 'a' .. 'z' 'A' .. 'Z' ]$ .

$[DIGIT]$         ::= $[ '0' .. '9' ]$ .

$[SPECIALS]$      ::= $[ '&'   ')' '+' ',' ';'  '|' ]$ .

$[STRINGCHARS]$   ::= $[ NUL .. '!' '#' .. DEL ]$ .

$[ILLEGALS]$      ::= $[ NUL .. SP '#' .. '%' '?' .. '@' '[' .. '^'
                         '{' '}' .. EOL EOC .. BAD ]$ .

$[COMMENTCHARS]$  ::= $[ NUL .. DEL  ]$ .

$[BASEARITH]$     ::= $[ 'A' 'B' 'C' 'D' 'E' 'F' ]$ .

$[CHARACTERLIT]$  ::= $[ '!' .. '~' ]$ .
```

10. APPENDIX II. EXAMPLE OF DIANA PRODUCTION.

10.1. APPENDIX IIa. ADA FRAGMENTS.

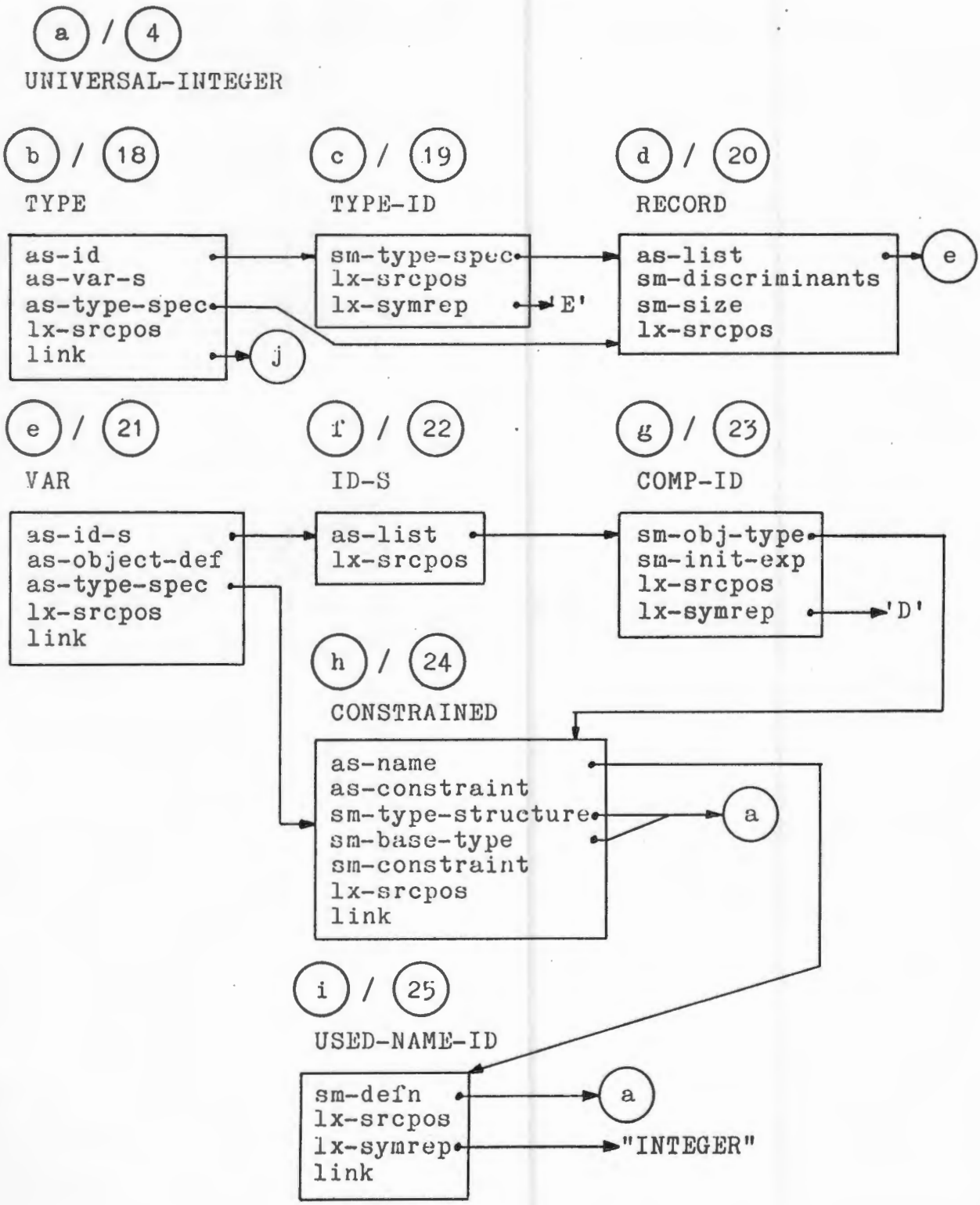
```
.
.
.
TYPE E IS RECORD
    D : INTEGER ;
END RECORD ;
C : E ;
A,B : INTEGER ;
F : ARRAY ( 1 .. 10 ) OF INTEGER;
BEGIN
A := 3;
B := 0;
WHILE B < 10 LOOP
  F(B) := (4*3+B)/A;
  B := B+A;
END LOOP ;
C.D := B ;
END
.
.
.
```

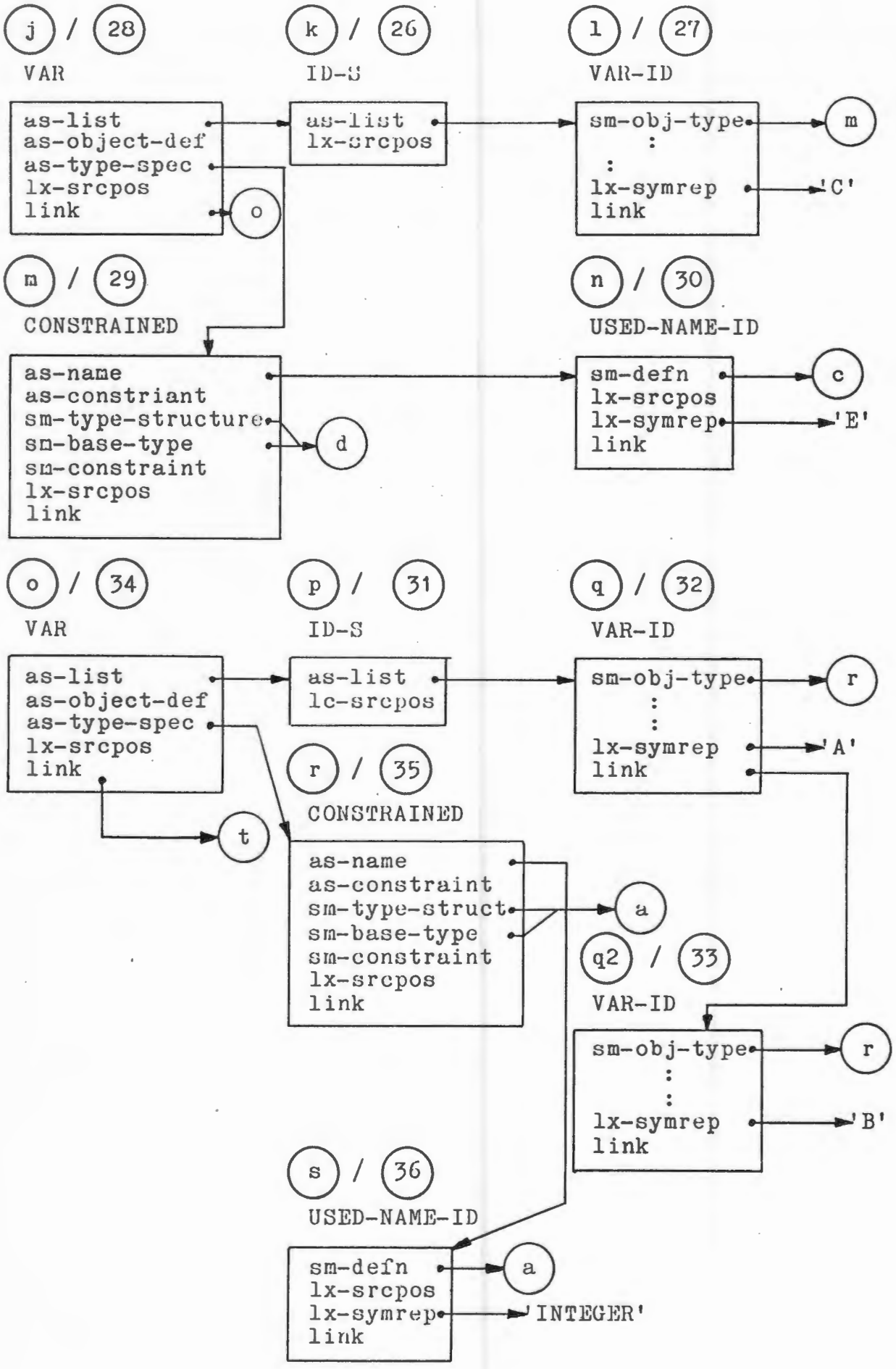
10.2. APPENDIX IIb. DIANA ABSTRACT SYNTAX TREE FRAGMENTS.

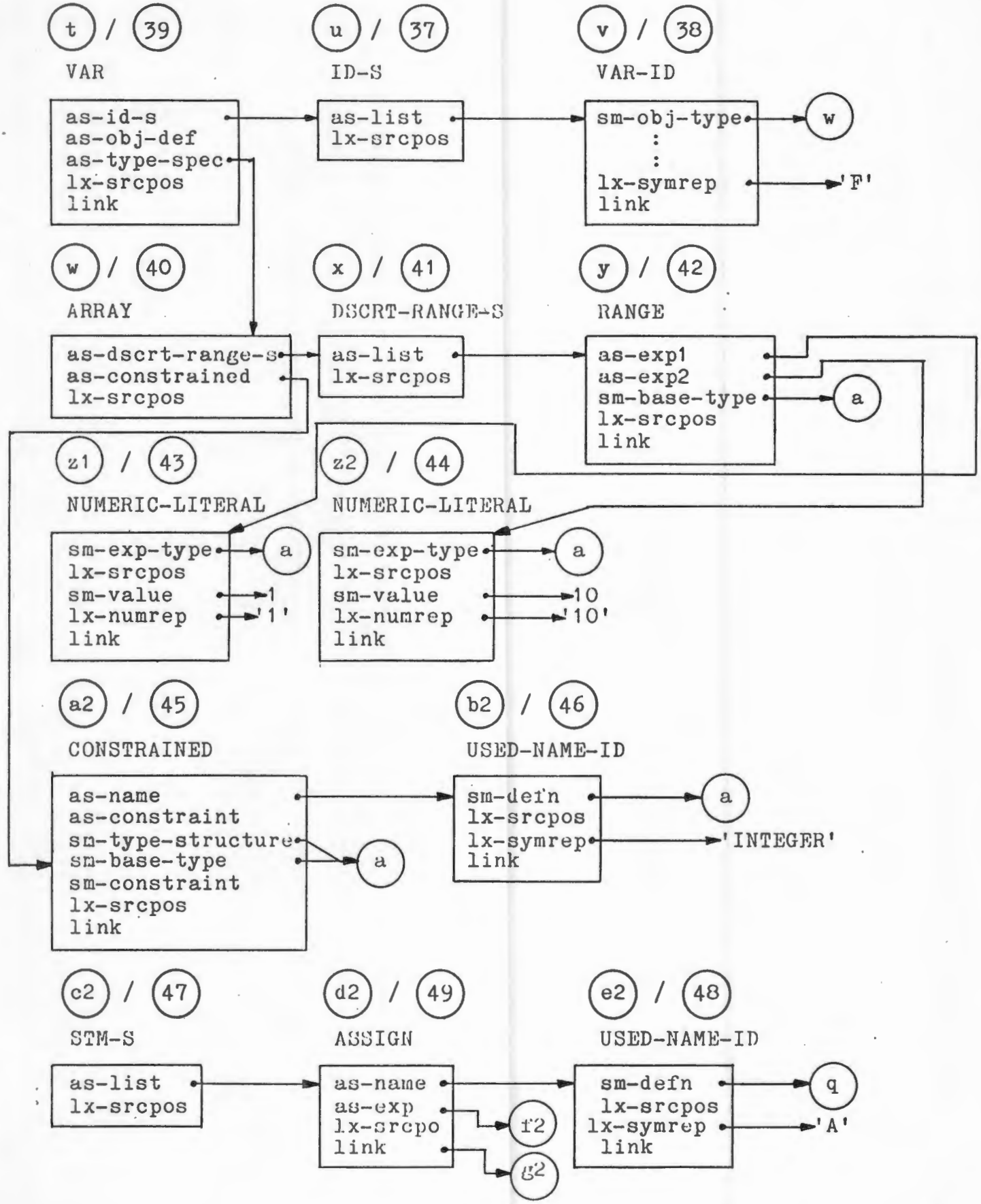
Key (a) : connector

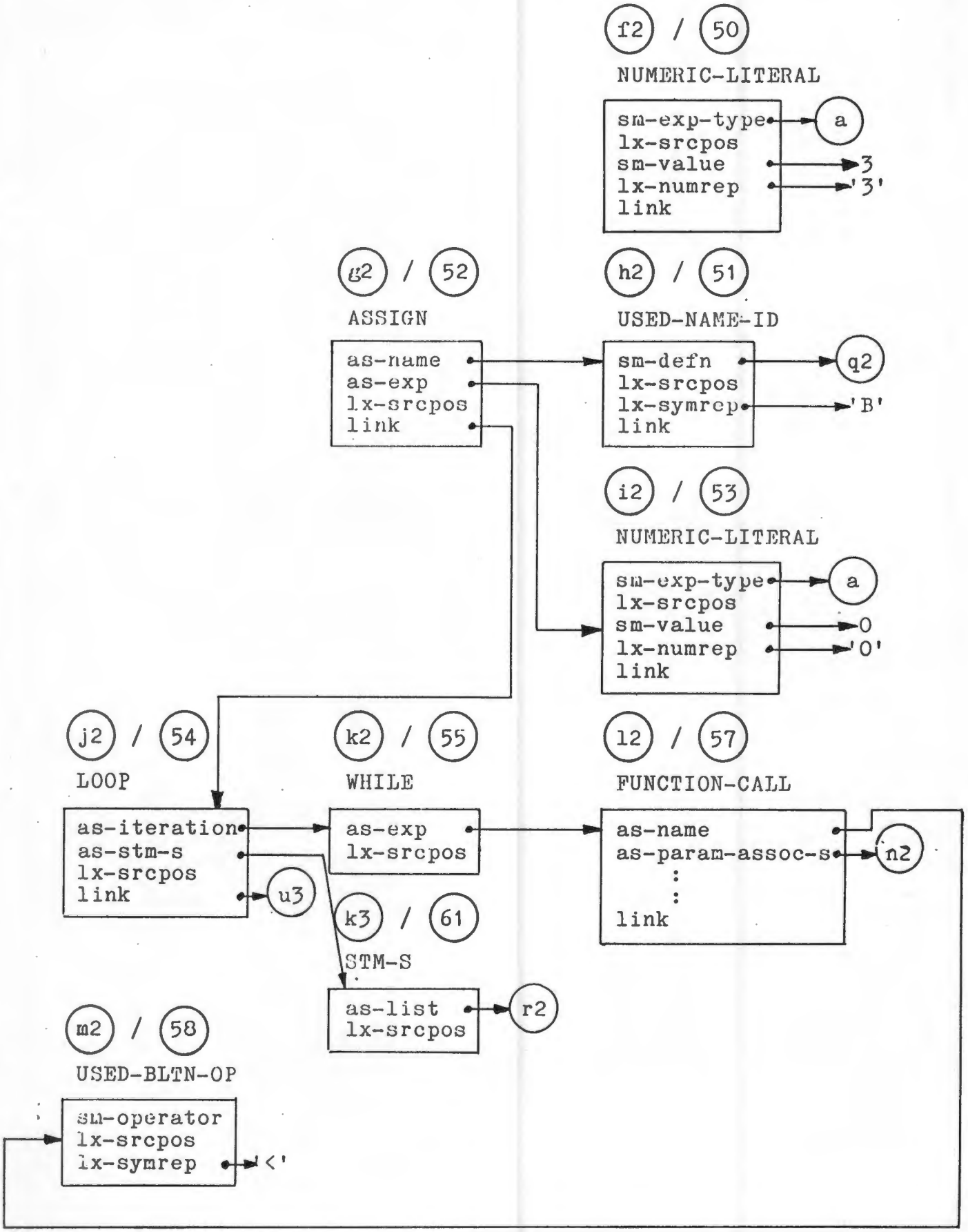
Note : Alphabetic node numbers - hand generated sequence nos.

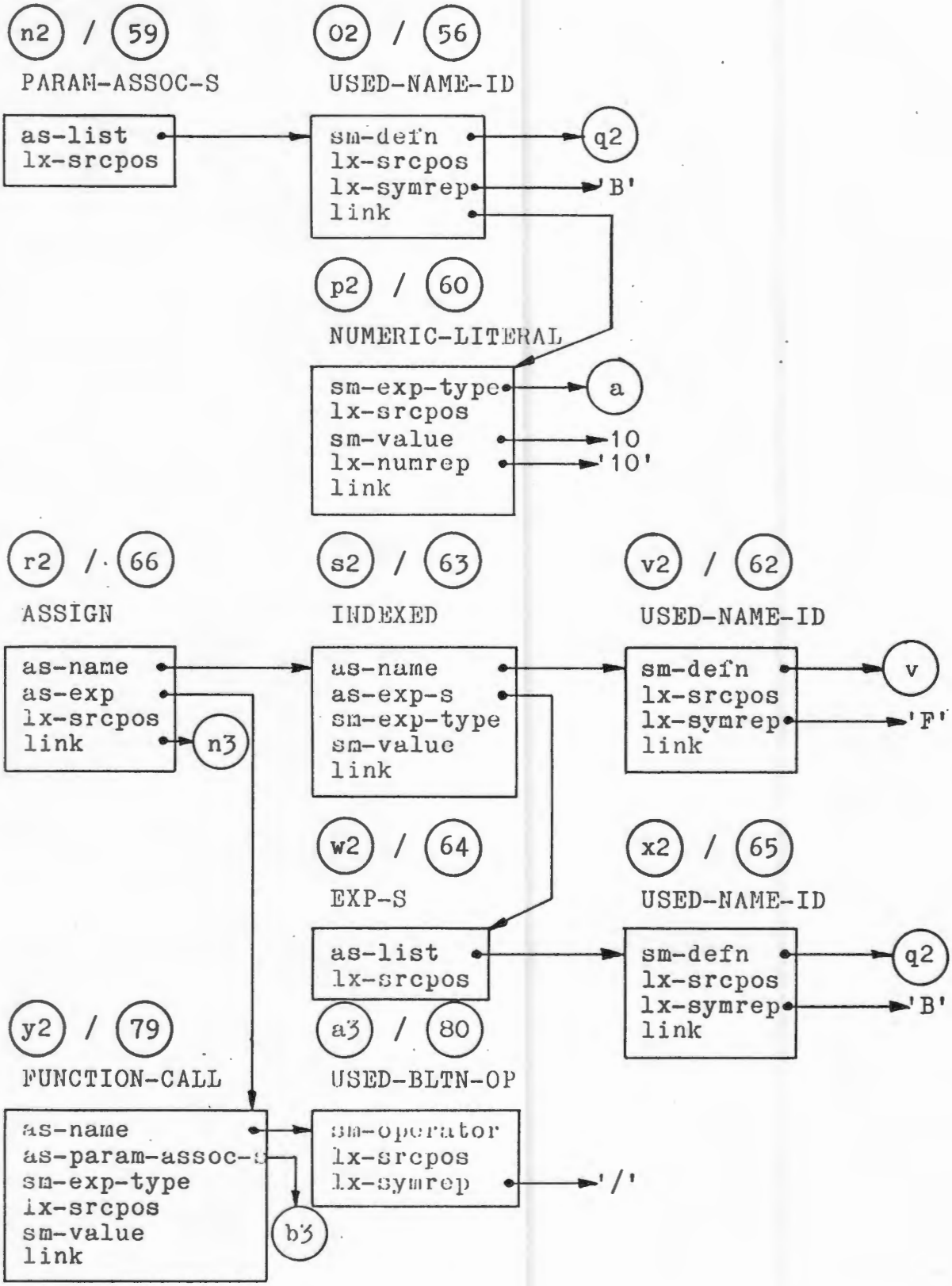
Numeric node numbers - program generated sequence nos.

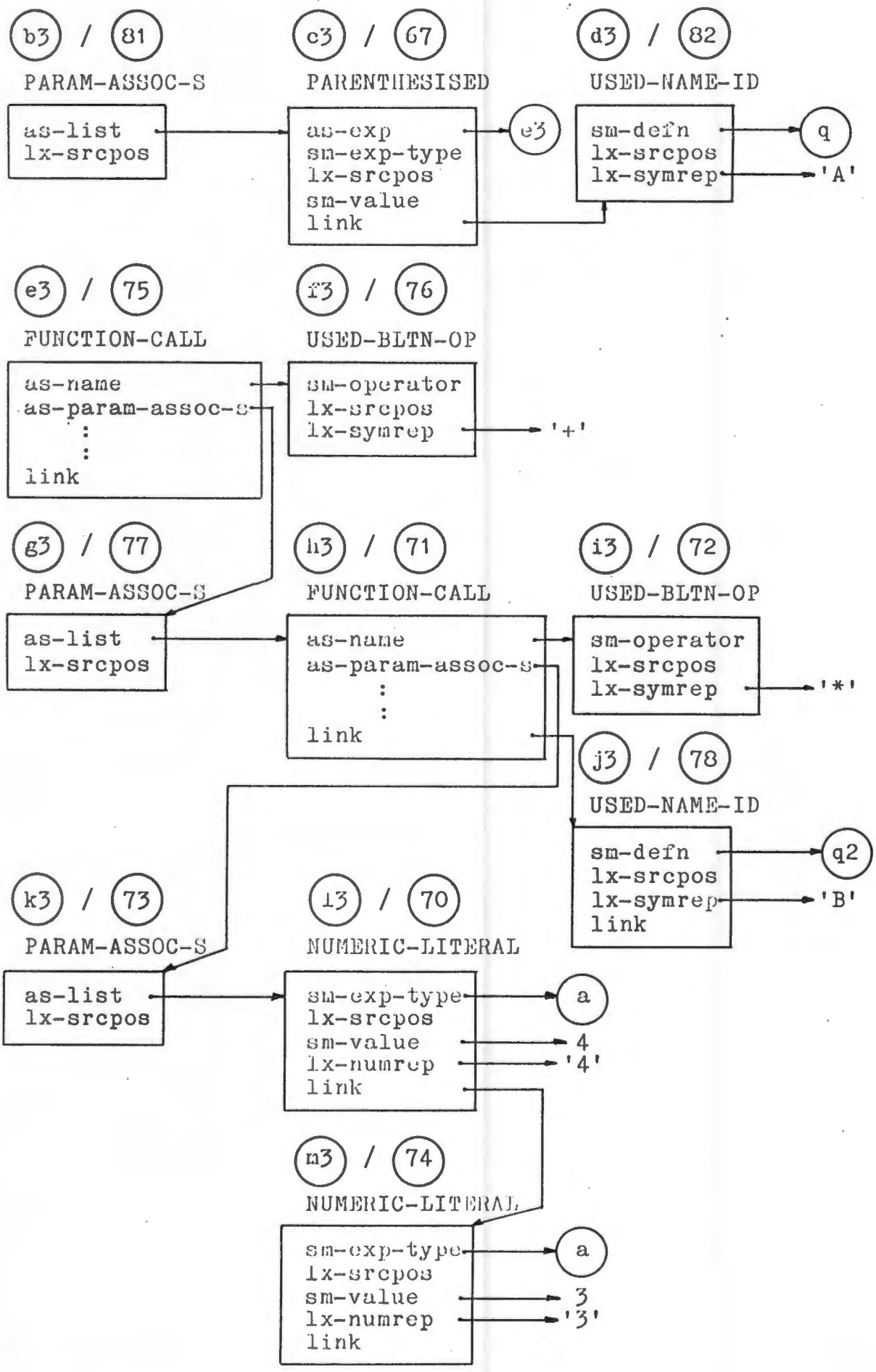


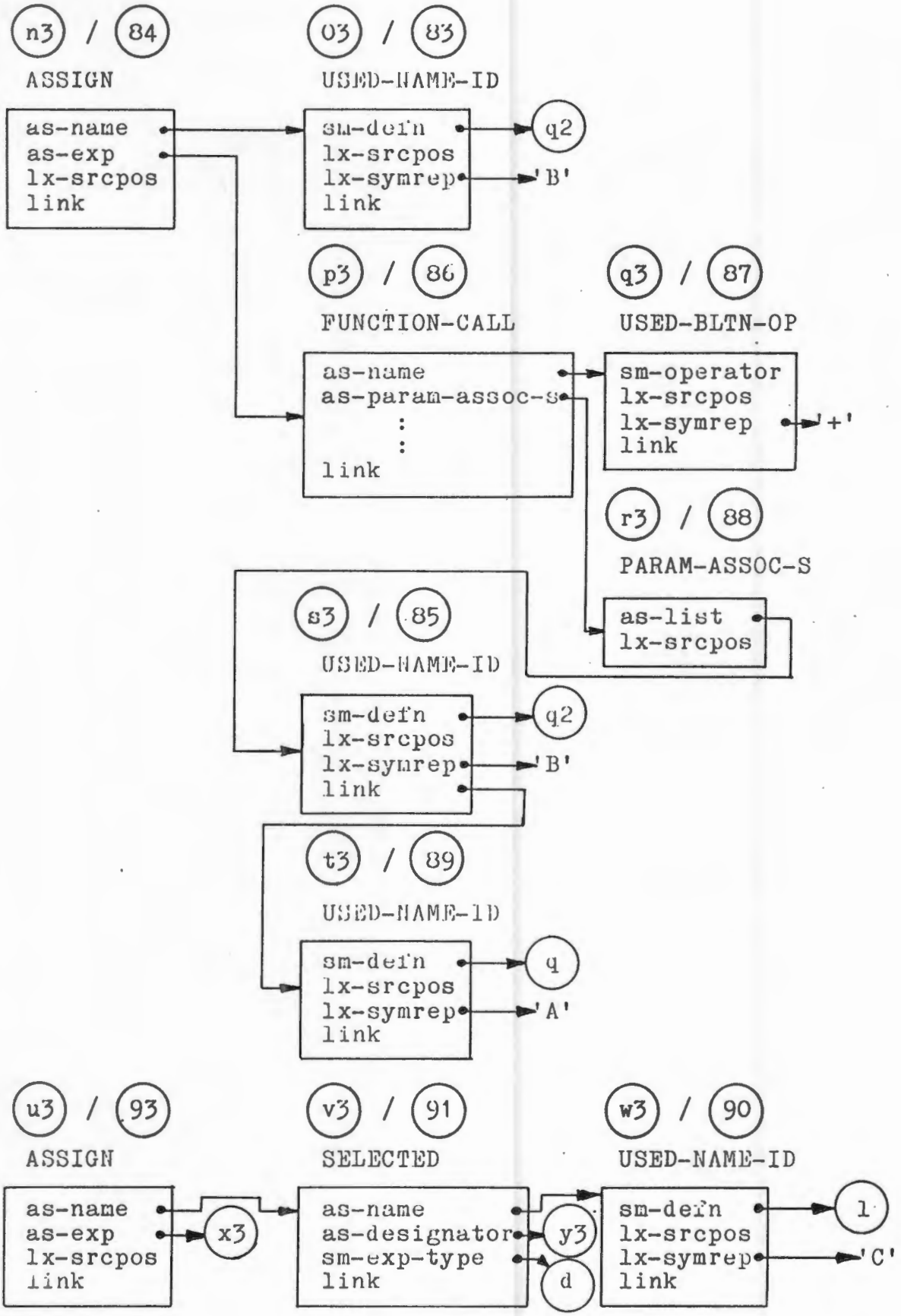

j / 28
VAR
as-list
as-object-def
as-type-spec
lx-srcpos
link
o
k / 26
ID-S
as-list
lx-srcpos
l / 27
VAR-ID
sm-obj-type
:
:
lx-symrep
link
m
'C'
m / 29
CONSTRAINED
as-name
as-constraint
sm-type-structure
sm-base-type
sm-constraint
lx-srcpos
link
d
n / 30
USED-NAME-ID
sm-defn
lx-srcpos
lx-symrep
link
c
'E'
o / 34
VAR
as-list
as-object-def
as-type-spec
lx-srcpos
link
t
p / 31
ID-S
as-list
lc-srcpos
q / 32
VAR-ID
sm-obj-type
:
:
lx-symrep
link
r
'A'
r / 35
CONSTRAINED
as-name
as-constraint
sm-type-struct
sm-base-type
sm-constraint
lx-srcpos
link
a
q2 / 33
VAR-ID
sm-obj-type
:
:
lx-symrep
link
r
'B'
s / 36
USED-NAME-ID
sm-defn
lx-srcpos
lx-symrep
link
a
'INTEGER'

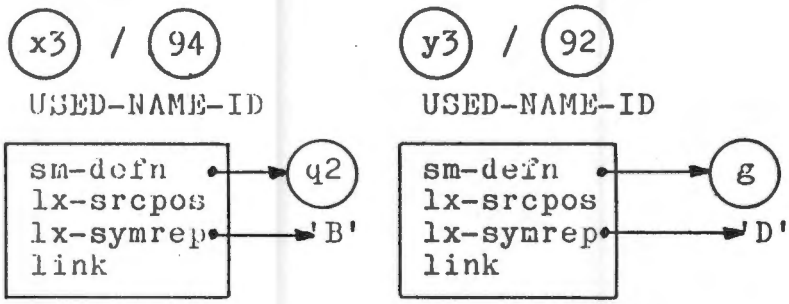

t / 39
VAR
as-id-s
as-obj-def
as-type-spec
lx-srcpos
link
u / 37
ID-S
as-list
lx-srcpos
v / 38
VAR-ID
sm-obj-type
:
:
lx-symrep
link
w
'F'
w / 40
ARRAY
as-dscrt-range-s
as-constrained
lx-srcpos
x / 41
DSCRT-RANGE-S
as-list
lx-srcpos
y / 42
RANGE
as-exp1
as-exp2
sm-base-type
lx-srcpos
link
a
z1 / 43
NUMERIC-LITERAL
sm-exp-type
lx-srcpos
sm-value
lx-numrep
link
a
1
'1'
z2 / 44
NUMERIC-LITERAL
sm-exp-type
lx-srcpos
sm-value
lx-numrep
link
a
10
'10'
a2 / 45
CONSTRAINED
as-name
as-constraint
sm-type-structure
sm-base-type
sm-constraint
lx-srcpos
link
a
b2 / 46
USED-NAME-ID
sm-defn
lx-srcpos
lx-symrep
link
a
'INTEGER'
c2 / 47
STM-S
as-list
lx-srcpos
d2 / 49
ASSIGN
as-name
as-exp
lx-srcpo
link
f2
g2
e2 / 48
USED-NAME-ID
sm-defn
lx-srcpos
lx-symrep
link
q
'A'


f2 / 50
NUMERIC-LITERAL
sm-exp-type
lx-srcpos
sm-value
lx-numrep
link
a
3
'3'
g2 / 52
ASSIGN
as-name
as-exp
lx-srcpos
link
h2 / 51
USED-NAME-ID
sm-defn
lx-srcpos
lx-symrep
link
q2
'B'
i2 / 53
NUMERIC-LITERAL
sm-exp-type
lx-srcpos
sm-value
lx-numrep
link
a
0
'0'
j2 / 54
LOOP
as-iteration
as-stm-s
lx-srcpos
link
u3
k2 / 55
WHILE
as-exp
lx-srcpos
l2 / 57
FUNCTION-CALL
as-name
as-param-assoc-s
link
n2
k3 / 61
STM-S
as-list
lx-srcpos
r2
m2 / 58
USED-BLTN-OP
sm-operator
lx-srcpos
lx-symrep
'<'


n2 / 59
PARAM-ASSOC-S
as-list
lx-srcpos
02 / 56
USED-NAME-ID
sm-defn
lx-srcpos
lx-symrep
link
q2
'B'
p2 / 60
NUMERIC-LITERAL
sm-exp-type
lx-srcpos
sm-value
lx-numrep
link
a
10
'10'
r2 / 66
ASSIGN
as-name
as-exp
lx-srcpos
link
n3
s2 / 63
INDEXED
as-name
as-exp-s
sm-exp-type
sm-value
link
v2 / 62
USED-NAME-ID
sm-defn
lx-srcpos
lx-symrep
link
v
'F'
w2 / 64
EXP-S
as-list
lx-srcpos
x2 / 65
USED-NAME-ID
sm-defn
lx-srcpos
lx-symrep
link
q2
'B'
y2 / 79
FUNCTION-CALL
as-name
as-param-assoc-s
sm-exp-type
lx-srcpos
sm-value
link
b3
a3 / 80
USED-BLTN-OP
sm-operator
lx-srcpos
lx-symrep
'/'


b3 / 81
PARAM-ASSOC-S
as-list
lx-srcpos
c3 / 67
PARENTHESISED
as-exp
sm-exp-type
lx-srcpos
sm-value
link
e3
d3 / 82
USED-NAME-ID
sm-defn
lx-srcpos
lx-symrep
q
'A'
e3 / 75
FUNCTION-CALL
as-name
as-param-assoc-s
:
:
link
f3 / 76
USED-BLTN-OP
sm-operator
lx-srcpos
lx-symrep
'+'
g3 / 77
PARAM-ASSOC-S
as-list
lx-srcpos
h3 / 71
FUNCTION-CALL
as-name
as-param-assoc-s
:
:
link
i3 / 72
USED-BLTN-OP
sm-operator
lx-srcpos
lx-symrep
'*'
j3 / 78
USED-NAME-ID
sm-defn
lx-srcpos
lx-symrep
link
q2
'B'
k3 / 73
PARAM-ASSOC-S
as-list
lx-srcpos
l3 / 70
NUMERIC-LITERAL
sm-exp-type
lx-srcpos
sm-value
lx-numrep
link
a
4
'4'
n3 / 74
NUMERIC-LITERAL
sm-exp-type
lx-srcpos
sm-value
lx-numrep
link
a
3
'3'


n3 / 84
ASSIGN
as-name
as-exp
lx-srcpos
link
03 / 83
USED-NAME-ID
sm-defn
lx-srcpos
lx-symrep
link
q2
'B'
p3 / 86
FUNCTION-CALL
as-name
as-param-assoc-s
:
:
link
q3 / 87
USED-BLTN-OP
sm-operator
lx-srcpos
lx-symrep
link
'+'
r3 / 88
PARAM-ASSOC-S
as-list
lx-srcpos
s3 / 85
USED-NAME-ID
sm-defn
lx-srcpos
lx-symrep
link
q2
'B'
t3 / 89
USED-NAME-ID
sm-defn
lx-srcpos
lx-symrep
link
q
'A'
u3 / 93
ASSIGN
as-name
as-exp
lx-srcpos
link
x3
v3 / 91
SELECTED
as-name
as-designator
sm-exp-type
link
y3
d
w3 / 90
USED-NAME-ID
sm-defn
lx-srcpos
lx-symrep
link
1
'C'


x3 / 94
USED-NAME-ID
sm-defn
lx-srcpos
lx-symrep
link
q2
'B'
y3 / 92
USED-NAME-ID
sm-defn
lx-srcpos
lx-symrep
link
g
'D'

10.3. APPENDIX IIc. DIANA REPRESENTATION.

DIANA REPRESENTATION PRODUCED BY THE UCT-FRONT END.

Key:

KND = box-kind.

SEQ = Sequence number of node found in ref(1).

REF2-REF6 = values stored in fields ref(2) - ref(6).

REF11-REF12 = values stored in fields ref(11) - ref(12).

VAL1 = value stored in the first value field.

REP1-REP2 = symbolic representation of the token.

The remaining fields in a node are not mentioned because they are not used in this example.

KND:	----	149	150	123	160	70	24	33	157	70
SEQ:		18	19	20	21	22	23	24	25	26
REF2:		19	20	21	22	23	24	25	4	27
REF3:										
REF4:		20			24			4		
REF5:								4		
REF6:										
REF11:										
REF12:		28								
VAL1:										
REP1:			E				D		INTE	
REP2:									GER	

KND:	161	160	33	157	70	161	161	160	33	157
SEQ:	27	28	29	30	31	32	33	34	35	36
REF2:	29	26	30	19	32	35	35	31	36	4
REF3:										
REF4:		29	20					35	4	
REF5:			20						4	
REF6:										
REF11:										
REF12:		34				33		39		
VAL1:										
REP1:	C			E		A	B		INTE	
REP2:									GER	

KND:	70	161	160	12	44	122	100	100	33	157
SEQ:	37	38	39	40	41	42	43	44	45	46
REF2:	38	40	37	41	42	43	4	4	46	4
REF3:				45		44				
REF4:			40			4			4	
REF5:									4	
REF6:										
REF11:										
REF12:										
VAL1:							1	10		
REP1:		F					1	10		INTE
REP2:										GER

KND:	134	157	13	100	157	13	100	88	167	157
SEQ:	47	48	49	50	51	52	53	54	55	56
REF2:	49	32	48	4	33	51	4	55	57	33
REF3:			50			53		61		
REF4:										
REF5:										
REF6:										
REF11:										
REF12:			52			54		93		60
VAL1:				3			0			
REP1:		A		3	B		0			B
REP2:										

KND:	62	138	109	100	134	157	78	53	157	13
SEQ:	57	58	59	60	61	62	63	64	65	66
REF2:	58		56	4	66	38	62	65	33	63
REF3:	59						64			79
REF4:										
REF5:										
REF6:										
REF11:										
REF12:										84
VAL1:				10						
REP1:		<		10		F			B	
REP2:										

KND:	111	100	62	138	109	100	62	138	109	157
SEQ:	67	70	71	72	73	74	75	76	77	78
REF2:	75	4	72		70	4	76		71	33
REF3:			73				77			
REF4:										
REF5:										
REF6:										
REF11:										
REF12:	82	74	78							
VAL1:		4				3				
REP1:		4		*		3		+		B
REP2:										

KND:	62	138	109	157	157	13	157	62	138	109
SEQ:	79	80	81	82	83	84	85	86	87	88
REF2:	80		67	32	33	83	33	87		85
REF3:	81					86		88		
REF4:										
REF5:										
REF6:										
REF11:										
REF12:							89			
VAL1:										
REP1:		/		A	B		B		+	
REP2:										

KND:	157	157	131	157	13	157
SEQ:	89	90	91	92	93	94
REF2:	32	27	90	23	91	33
REF3:			92		94	
REF4:			20			
REF5:						
REF6:						
REF11:						
REF12:						
VAL1:						
REP1:	A	C		D		B
REP2:						

11. APPENDIX III. STRUCTURE OF DIANA NODES.

Note:

1) reference(1) is always the sequence number of the node, i.e. node generation number. It is not mentioned in this appendix.

2) The attribute "link" does not appear in the manual, but is used to link the nodes forming a list.

NODE NAME	TYPE NUMBER	ATTRIBUTES	REFERENCE IN RECORD.
ABORT	1		not implemented
ACCEPT	2		not implemented
ACCESS	3	as-constrained	reference(2)
		sm-size	reference(3)
		sm-storage-size	reference(4)
		lx-srcpos	reference(5)
		cd-impl-size	value(1)
			boolean(1)
		cd-alignment	value(2)
			boolean(2)
		sm-controlled	boolean(5)
ADDRESS	4		not implemented
AGGREGATE	5	as-list	reference(2)
		sm-exp-type	reference(3)
		sm-constraint	reference(4)
		lx-srcpos	reference(5)
		link	reference(12)
		sm-value	value(1)
			boolean(1)
ALL	6	as-name	reference(2)
		sm-exp-type	reference(4)
		lx-srcpos	reference(5)
		link	reference(12)
		sm-value	value(1)
			boolean(1)

NODE NAME	TYPE NUMBER	ATTRIBUTES	REFERENCE IN RECORD.
ALLOCATOR	7	as-name	reference(2)
		as-access-constraint	reference(3)
		sm-exp-type	reference(4)
		lx-srcpos	reference(5)
		link	reference(12)
		sm-value	value(1)
			boolean(1)
ALTERNATIVE	8	as-choice-s	reference(2)
		as-stm-s	reference(3)
		lx-srcpos	reference(4)
		link	reference(12)
ALTERNATIVE-S	9	as-list	reference(2)
		lx-srcpos	reference(3)
AND-THEN	10	lx-srcpos	reference(2)
ARGUMENT-ID	11		not implemented
ARRAY	12	as-dscrt-range-s	reference(2)
		as-constrained	reference(3)
		sm-size	reference(4)
		lx-srcpos	reference(5)
		cd-impl-size	value(1)
			boolean(1)
		cd-alignment	value(2)
			boolean(2)
		sm-packing	boolean(5)
ASSIGN	13	as-name	reference(2)
		as-exp	reference(3)
		lx-srcpos	reference(4)
		link	reference(12)
ASSOC	14	as-id	reference(2)
		as-actual	reference(3)
		lx-srcpos	reference(4)
		link	reference(12)
ATTR-ID	15		not implemented
ATTRIBUTE	16		not implemented
ATTRIBUTE-CALL	17		not implemented

NODE NAME	TYPE NUMBER	ATTRIBUTES	REFERENCE IN RECORD.
BINARY	18	as-exp1	reference(2)
		as-binary-op	reference(3)
		as-exp2	reference(4)
		sm-exp-type	reference(5)
		lx-srcpos	reference(6)
		link	reference(12)
		sm-value	value(1)
			boolean(1)
BLOCK	19	as-item-s	reference(2)
		as-stm-s	reference(3)
		as-alternative-s	reference(4)
		lx-srcpos	reference(5)
		link	reference(12)
BOX	20		not implemented
CASE	21	as-exp	reference(2)
		as-alternative-s	reference(3)
		lx-srcpos	reference(4)
CHOICE-S	22	as-list-s	reference(2)
		lx-srcpos	reference(3)
CODE	23	as-exp	reference(2)
		lx-srcpos	reference(3)
COMP-ID	24	sm-obj-type	reference(2)
		sm-init-exp	reference(3)
		lx-srcpos	reference(4)
		link	reference(12)
		cd-position	value(1)
			boolean(1)
		cd-first-bit	value(2)
			boolean(2)
		cd-last-bit	value(3)
			boolean(3)
		lx-symrep	rep(1) - rep(4)
COMP-REP	25		not implemented
COMP-REP-S	26		not implemented
COMP-UNIT	27	as-pragma-s	reference(2)
		as-context	reference(3)
		as-unit-body	reference(4)
		lx-srcpos	reference(5)
		link	reference(12)

NODE NAME	TYPE NUMBER	ATTRIBUTES	REFERENCE IN RECORD.
COMPILATION	28	as-list	reference(2)
		lx-srcpos	reference(3)
COND-CLAUSE	29	as-exp-void	reference(2)
		as-stm-s	reference(3)
		lx-srcpos	reference(4)
		link	reference(12)
COND-ENTRY	30		not implemented
CONST-ID	31	sm-obj-type	reference(2)
		sm-obj-def	reference(3)
		sm-address	reference(4)
		lx-srcpos	reference(5)
		link	reference(12)
		lx-symrep	rep(1) - rep(4)
CONSTANT	32	as-id-s	reference(2)
		as-object-def	reference(3)
		as-type-spec	reference(4)
		lx-srcpos	reference(5)
		link	reference(12)
CONSTRAINED	33	as-name	reference(2)
		as-constraint	reference(3)
		sm-type-struct	reference(4)
		sm-base-type	reference(5)
		sm-constraint	reference(6)
		lx-srcpos	reference(7)
		link	reference(12)
		cd-impl-size	value(1)
			boolean(1)
		cd-alignment	value(2)
			boolean(2)
CONTEXT	34	as-list	reference(2)
		lx-srcpos	reference(3)
CONVERSION	35	as-name	reference(2)
		as-exp	reference(3)
		sm-exp-type	reference(4)
		lx-srcpos	reference(5)
		link	reference(12)
		sm-value	value(1)
			boolean(1)
DECL-REP-S	36		not implemented
DECL-S	37	as-list	reference(2)
		lx-srcpos	reference(3)

NODE NAME	TYPE NUMBER	ATTRIBUTES	REFERENCE IN RECORD.
DEF-CHAR	38	sm-obj-type	reference(2)
		lx-srcpos	reference(3)
		sm-pos	value(1)
			boolean(1)
		sm-rep	value(2)
			boolean(2)
		lx-symrep	rep(1) - rep(4)
DEF-OP	39	sm-spec	reference(2)
		sm-body	reference(3)
		sm-location	reference(4)
		lx-srcpos	reference(5)
		lx-symrep	rep(1) - rep(4)
DELAY	40		not implemented
DERIVED	41	as-constrained	reference(2)
		sm-size	reference(3)
		sm-storage-size	reference(4)
		lx-srcpos	reference(5)
		cd-impl-size	value(1)
			boolean(1)
		cd-alignment	value(2)
			boolean(2)
		sm-actual-delta	value(3)
			boolean(3)
		sm-packing	boolean(5)
		sm-controlled	boolean(6)
DISCRIMANT-AGGREGATE	42	as-list	reference(2)
		lx-srcpos	reference(3)
DSCRMT-ID	43	sm-obj-type	reference(2)
		sm-init-exp	reference(3)
		lx-srcpos	reference(4)
		cd-position	value(1)
			boolean(1)
		cd-first-bit	value(2)
			boolean(2)
		cd-last-bit	value(3)
			boolean(3)
		lx-symrep	rep(1) - rep(4)
DSCRT-RANGE-S	44	as-list	reference(2)
		lx-srcpos	reference(3)
ENTRY	45		not implemented
ENTRY-CALL	46		not implemented

NODE NAME	TYPE NUMBER	ATTRIBUTES	REFERENCE IN RECORD.
ENTRY-ID	47	sm-spec	reference(2)
		sm-address	reference(3)
		lx-srcpos	reference(4)
		lx-symrep	rep(1) - rep(4)
ENUM-ID	48	sm-obj-type	reference(2)
		lx-srcpos	reference(3)
		sm-pos	value(1)
			boolean(1)
		sm-rep	value(2)
			boolean(2)
		lx-symrep	rep(1) - rep(4)
ENUM-LITERAL-S	49	as-list	reference(2)
		sm-size	reference(3)
		lx-srcpos	reference(4)
		cd-impl-size	value(1)
			boolean(1)
		cd-alignment	value(2)
			boolean(2)
EXCEPTION	50		not implemented
EXCEPTION-ID	51	sm-exception-def	reference(2)
		lx-srcpos	reference(3)
		lx-symrep	rep(1) - rep(4)
EXIT	52	as-name-void	reference(2)
		as-exp-void	reference(3)
		sm-stm	reference(4)
		lx-srcpos	reference(5)
EXP-S	53	as-list	reference(2)
		lx-srcpos	reference(3)
FIXED	54	as-exp	reference(2)
		as-range-void	reference(3)
		sm-size	reference(4)
		lx-srcpos	reference(5)
		cd-impl-size	value(1)
			boolean(1)
		cd-alignment	value(2)
			boolean(2)
		sm-actual-delta	value(3)
			boolean(3)
		sm-bits	value(4)
			boolean(4)

NODE NAME	TYPE NUMBER	ATTRIBUTES	REFERENCE IN RECORD.
FLOAT	55	as-exp	reference(2)
		as-range-void	reference(3)
		sm-type-struct	reference(4)
		sm-size	reference(5)
		lx-srcpos	reference(6)
		cd-impl-size	value(1)
			boolean(1)
		cd-alignment	value(2)
			boolean(2)
FOR	56	as-id	reference(2)
		as-dscrt-range	reference(3)
		lx-srcpos	reference(4)
FORMAL-DSCRT	57		not implemented
FORMAL-FIXED	58		not implemented
FORMAL-FLOAT	59		not implemented
FORMAL-INTEGER	60		not implemented
FUNCTION	61	as-param-s	reference(2)
		as-constrained-void	reference(3)
		lx-srcpos	reference(4)
FUNCTION-CALL	62	as-name	reference(2)
		as-param-assoc-s	reference(3)
		sm-exp-type	reference(5)
		lx-srcpos	reference(6)
		sm-value	value(1)
			boolean(1)
FUNCTION-ID	63	sm-spec	reference(2)
		sm-body	reference(3)
		sm-location	reference(4)
		lx-srcpos	reference(5)
		lx-symrep	rep(1) - rep(4)
GENERIC	64		not implemented
GENERIC-ASSOC-S	65		not implemented
GENERIC-ID	66	sm-generic-param-s	reference(2)
		sm-spec	reference(3)
		sm-body	reference(4)
		lx-srcpos	reference(5)
		lx-symrep	rep(1) - rep(4)
GENERIC-OP	67		not implemented
GENERIC-PARAM-S	68		not implemented

NODE NAME	TYPE NUMBER	ATTRIBUTES	REFERENCE IN RECORD.
GO TO	69	as-name	reference(2)
		lx-srcpos	reference(3)
		link	reference(12)
ID-S	70	as-list	reference(2)
		lx-srcpos	reference(3)
IF	71	as-list	reference(2)
		lx-srcpos	reference(3)
		link	reference(12)
IN	72	as-id-s	reference(2)
		as-type-spec	reference(3)
		as-exp-void	reference(4)
		lx-srcpos	reference(5)
		link	reference(12)
IN-ID	73	sm-obj-type	reference(2)
		sm-init-exp	reference(3)
		lx-srcpos	reference(5)
		link	reference(12)
		lx-symrep	rep(1) - rep(4)
IN-OP	74	lx-srcpos	reference(2)
IN-OUT	75	as-id-s	reference(2)
		as-type-spec	reference(3)
		as-exp-void	reference(4)
		lx-srcpos	reference(5)
		link	reference(12)
IN-OUT-ID	76	sm-obj-type	reference(2)
		lx-srcpos	reference(3)
		link	reference(12)
		lx-symrep	rep(1) - rep(4)
INDEX	77	as-name	reference(2)
		lx-srcpos	reference(3)
		link	reference(12)
INDEXED	78	as-name	reference(2)
		as-exp-s	reference(3)
		sm-exp-type	reference(4)
		link	reference(12)
		sm-value	value(1)
			boolean(1)

NODE NAME	TYPE NUMBER	ATTRIBUTES	REFERENCE IN RECORD.
INNER-RECORD	79	as-list	reference(2)
		lx-srcpos	reference(3)
INSTANTIATION	80		not implemented
INTEGER	81	as-range	reference(2)
		sm-type-struct	reference(3)
		sm-size	reference(4)
		lx-srcpos	reference(5)
		cd-impl-size	value(1)
			boolean(1)
		cd-alignment	value(2)
			boolean(2)
ITEM-S	82	as-list	reference(2)
		lx-srcpos	reference(3)
ITERATION-ID	83	sm-obj-type	reference(2)
		lx-srcpos	reference(3)
		lx-symrep	rep(1) - rep(4)
L-PRIVATE	84		not implemented
L-PRIVATE-TYPE-ID	85		not implemented
LABEL-ID	86	sm-stm	reference(2)
		lx-srcpos	reference(3)
		link	reference(12)
		lx-symrep	rep(1) - rep(4)
LABELLED	87	as-id-s	reference(2)
		as-stm	reference(3)
		lx-srcpos	reference(4)
		link	reference(12)
LOOP	88	as-iteration	reference(2)
		as-stm-s	reference(3)
		lx-srcpos	reference(4)
		link	reference(12)
MEMBERSHIP	89	as-exp	reference(2)
		as-membership-op	reference(3)
		as-type-range	reference(4)
		sm-exp-type	reference(5)
		lx-srcpos	reference(6)
		link	reference(12)
		sm-value	value(1)
			boolean(1)
NAME-S	90	as-list	reference(2)
		lx-srcpos	reference(3)

NODE NAME	TYPE NUMBER	ATTRIBUTES	REFERENCE IN RECORD.
NAMED	91	as-choice-s	reference(2)
		as-exp	reference(3)
		lx-srcpos	reference(4)
		link	reference(12)
NAMED-STM	92	as-id	reference(2)
		as-stm	reference(3)
		lx-srcpos	reference(4)
		link	reference(12)
NO-DEFAULT	93		not implemented
NOT-IN	94	lx-srcpos	reference(2)
NULL-ACCESS	95	sm-exp-type	reference(2)
		lx-srcpos	reference(3)
		sm-value	value(1)
			boolean(1)
NULL-COMP	96	lx-srcpos	reference(2)
NULL-STM	97	lx-srcpos	reference(2)
NUMBER	98	as-id-s	reference(2)
		as-exp	reference(3)
		lx-srcpos	reference(5)
		link	reference(12)
NUMBER-ID	99	sm-obj-type	reference(2)
		as-init-exp	reference(3)
		lx-srcpos	reference(4)
		link	reference(12)
		lx-symrep	rep(1) - rep(4)
NUMERIC-LITERAL	100	sm-exp-type	reference(2)
		lx-srcpos	reference(2)
		link	reference(12)
		sm-value	value(1)
			boolean(1)
		lx-numrep	rep(1) - rep(4)
OR-ELSE	101	lx-srcpos	reference(2)
OTHERS	102	lx-srcpos	reference(2)
		link	reference(12)

NODE NAME	TYPE NUMBER	ATTRIBUTES	REFERENCE IN RECORD.
OUT	103	as-id-s	reference(2)
		as-type-spec	reference(3)
		as-exp-void	reference(4)
		lx-srcpos	reference(5)
		link	reference(12)
OUT-ID	104	sm-obj-type	reference(2)
		lx-srcpos	reference(3)
		link	reference(12)
		lx-symrep	rep(1) - rep(4)
PACKAGE-BODY	105		not implemented
PACKAGE-DECL	106		not implemented
PACKAGE-ID	107	sm-spec	reference(2)
		sm-body	reference(3)
		sm-address	reference(4)
		lx-srcpos	reference(5)
		lx-symrep	rep(1) - rep(4)
PACKAGE-SPEC	108		not implemented
PARAM-ASSOC-S	109	as-list	reference(2)
		lx-srcpos	reference(3)
PARAM-S	110	as-list	reference(2)
		lx-srcpos	reference(3)
PARENTHESISED	111	as-exp	reference(2)
		sm-exp-type	reference(3)
		lx-srcpos	reference(4)
		link	reference(12)
		sm-value	value(1)
			boolean(1)
PRAGMA	112	as-id	reference(2)
		as-param-assoc-s	reference(3)
		lx-srcpos	reference(4)
		link	reference(12)
PRAGMA-ID	113	as-list	reference(2)
		lx-srcpos	reference(3)
PRAGMA-S	114	as-list	reference(2)
		lx-srcpos	reference(3)
PRIVATE	115		not implemented

NODE NAME	TYPE NUMBER	ATTRIBUTES	REFERENCE IN RECORD.
PRIVATE-TYPE-ID	116		not implemented
PROC-ID	117	sm-spec	reference(2)
		sm-body	reference(3)
		sm-location	reference(4)
		lx-srcpos	reference(5)
		lx-symrep	rep(1) - rep(4)
PROCEDURE	118	as-param-s	reference(2)
		lx-srcpos	reference(3)
PROCEDURE-CALL	119	as-name	reference(2)
		as-param-assoc-s	reference(3)
		lx-srcpos	reference(4)
		link	reference(12)
QUALIFIED	120	as-name	reference(2)
		as-exp	reference(3)
		sm-exp-type	reference(4)
		lx-srcpos	reference(5)
		link	reference(12)
		sm-value	value(1)
			boolean(1)
RAISE	121		not implemented
RANGE	122	as-exp2	reference(1)
		as-exp2	reference(3)
		sm-base-type	reference(4)
		lx-srcpos	reference(5)
		link	reference(12)
RECORD	123	as-list	reference(2)
		sm-discriminants	reference(3)
		sm-size	reference(4)
		lx-srcpos	reference(5)
		cd-impl-size	value(1)
			boolean(1)
		cd-alignment	value(2)
			boolean(2)
		sm-packing	boolean(5)
RECORD-REP	124		not implemented
RENAME	125		not implemented
RETURN	126	as-exp-void	reference(2)
		lx-srcpos	reference(3)
		link	reference(12)

NODE NAME	TYPE NUMBER	ATTRIBUTES	REFERENCE IN RECORD.
REVERSE	127	as-id	reference(2)
		as-dscrt-range	reference(3)
		lx-srcpos	reference(4)
SELECT	128		not implemented
SELECT-CLAUSE	129		not implemented
SELECT-CLAUSE-S	130		not implemented
SELECTED	131	as-name	reference(2)
		as-designator	reference(3)
		sm-exp-type	reference(4)
		lx-srcpos	reference(5)
		link	reference(12)
		sm-value	value(1)
			boolean(1)
SIMPLE-REP	132		not implemented
SLICE	133	as-name	reference(2)
		as-dscrt-range	reference(3)
		sm-exp-type	reference(4)
		sm-constraint	reference(5)
		lx-srcpos	reference(6)
		link	reference(12)
		sm-value	value(1)
			boolean(1)
STM-S	134	as-list	reference(2)
		lx-srcpos	reference(3)
STRING-LITERAL	135	sm-exp-type	reference(2)
		sm-constraint	reference(3)
		lx-srcpos	reference(4)
		link	reference(12)
		sm-value	value(1)
			boolean(1)
		lx-symrep	rep(1) - rep(4)
STUB	136		not implemented
SUBPROGRAM-BODY	137	as-designator	reference(2)
		as-header	reference(3)
		as-block-stub	reference(4)
		lx-srcpos	reference(5)
		link	reference(12)

NODE NAME	TYPE NUMBER	ATTRIBUTES	REFERENCE IN RECORD.
USED-BLTN-OP	138	sm-operator	reference(2)
		lx-srcpos	reference(3)
		lx-symrep	rep(1) - rep(4)
SUBPROGRAM-DECL	139		not implemented
SUBTYPE	140	as-id	reference(2)
		as-constrained	reference(3)
		lx-srcpos	reference(4)
		link	reference(12)
SUBTYPE-ID	141	sm-type-spec	reference(2)
		lx-srcpos	reference(3)
		lx-symrep	rep(1) - rep(4)
SUBUNIT	142		not implemented
TASK-BODY	143		not implemented
TASK-BODY-ID	144	sm-type-spec	reference(2)
		sm-body	reference(3)
		lx-srcpos	reference(4)
		lx-symrep	rep(1) - rep(4)
TASK-DECL	145		not implemented
TASK-SPEC	146		not implemented
TERMINATE	147		not implemented
TIMED-ENTRY	148		not implemented
TYPE	149	as-id	reference(2)
		as-var-s	reference(3)
		as-type-spec	reference(4)
		lx-srcpos	reference(5)
		link	reference(12)
TYPE-ID	150	sm-type-spec	reference(2)
		lx-srcpos	reference(3)
		lx-symrep	rep(1) - rep(4)
UNIVERSAL-FIXED	151	denoted by a 2 as the attribute value	
UNIVERSAL-INTEGER	152	denoted by a 4 as the attribute value	
UNIVERSAL-REAL	153	denoted by a 3 as the attribute value	

NODE NAME	TYPE NUMBER	ATTRIBUTES	REFERENCE IN RECORD.
USE	154	as-list	reference(2)
		lx-srcpos	reference(3)
		link	reference(12)
USED-BLT-ID	155	sm-operator	reference(2)
		lx-srcpos	reference(3)
		lx-symrep	rep(1) - rep(4)
USED-CHAR	156	sm-defn	reference(2)
		sm-exp-type	reference(3)
		lx-srcpos	reference(4)
		sm-value	value(1)
			boolean(1)
USED-NAME-ID	157	sm-defn	reference(2)
		lx-srcpos	reference(3)
		link	reference(12)
		lx-symrep	rep(1) - rep(4)
USED-OBJECT-ID	158	sm-defn	reference(2)
		sm-exp-type	reference(3)
		lx-srcpos	reference(4)
		link	reference(12)
		sm-value	value(1)
			boolean(1)
		lx-symrep	rep(1) - rep(4)
USED-OP	159	sm-defn	reference(2)
		link	reference(12)
		lx-symrep	rep(1) - rep(4)
VAR	160	as-id-s	reference(2)
		as-object-def	reference(3)
		as-type-spec	reference(4)
		lx-srcpos	reference(5)
		link	reference(12)
VAR-ID	161	as-obj-type	reference(2)
		sm-obj-def	reference(3)
		sm-address	reference(4)
		lx-srcpos	reference(5)
		link	reference(12)
		lx-symrep	rep(1) - rep(4)
VAR-S	162	as-list	reference(2)
		lx-srcpos	reference(3)
VARIANT	163	as-choice-s	reference(2)
		as-record	reference(3)
		lx-srcpos	reference(4)

NODE NAME	TYPE NUMBER	ATTRIBUTES	REFERENCE IN RECORD:
VARIANT-PART	164	as-name	reference(2)
		as-variant-s	reference(3)
		lx-srcpos	reference(4)
		link	reference(12)
VARIANT-S	165	as-list	reference(2)
		lx-srcpos	reference(3)
VOID	166		
WHILE	167	as-exp	reference(2)
		lx-srcpos	reference(3)
WITH	168	as-list	reference(2)
		lx-srcpos	reference(3)
		link	reference(12)
CONTINUATION	512	link	reference(11)

12. APPENDIX IV. DIANA PRODUCED FOR EXAMPLES IN CHAPTER 7.

12.1. APPENDIX IVa.

```
   28    11    12     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
  118    13     0     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
  117    14    13    16     0     0     0     0     0     0     0
0      0.00000     .00000    00TSORT
  137    15    14    13    16     0     0     0     0     0     0
0      0.00000     .00000    00
   82    17    18     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
  149    18    19     0     0     0     0     0     0     0     0
0     20.00000     .00000    00
  150    19     0     0     0     0     0     0     0     0     0
0      0.00000     .00000    00LEADER
  149    20    21     0     0     0     0     0     0     0     0
0     22.00000     .00000    00
  150    21     0     0     0     0     0     0     0     0     0
0      0.00000     .00000    00TRAILER
  149    22    23     0    24     0     0     0     0     0     0
0     27.00000     .00000    00
  150    23    24     0     0     0     0     0     0     0     0
0      0.00000     .00000    00LREF
    3    24    25     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
   33    25    26     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
  157    26    19     0     0     0     0     0     0     0     0
0      0.00000     .00000    00LEADER
  149    27    28     0    29     0     0     0     0     0     0
0     32.00000     .00000    00
  150    28    29     0     0     0     0     0     0     0     0
0      0.00000     .00000    00TREF
    3    29    30     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
   33    30    31     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
  157    31    21     0     0     0     0     0     0     0     0
0      0.00000     .00000    00TRAILER
  123    34    35     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
   70    36    37     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
   24    37    38    40     0     0     0     0     0     0     0
0      0.00000     .00000    00KEY
   33    38    39     0     4     4     0     0     0     0     0
0      0.00000     .00000    00
  157    39     4     0     0     0     0     0     0     0     0
0      0.00000     .00000    00INTEGER
  100    40     4     0     0     0     0     0     0     0     0
0      0.00000     .00000    100
  160    35    36    40    38     0     0     0     0     0     0
```

```
0    41.00000     .00000    00
   70     42    43     0     0     0     0     0     0     0     0
0       0.00000     .00000    00
   24     43    44    46     0     0     0     0     0     0     0
0       0.00000     .00000    00COUNT
   33     44    45     0     4     4     0     0     0     0     0
0       0.00000     .00000    00
  157     45     4     0     0     0     0     0     0     0     0
0       0.00000     .00000    00INTEGER
  100     46     4     0     0     0     0     0     0     0     0
0       0.00000     .00000    100
  160     41    42    46    44     0     0     0     0     0     0
0    47.00000     .00000    00
   70     48    49     0     0     0     0     0     0     0     0
0       0.00000     .00000    00
   24     49    50    52     0     0     0     0     0     0     0
0       0.00000     .00000    00TRAIL
   33     50    51     0     0    29     0     0     0     0     0
0       0.00000     .00000    00
  157     51    28     0     0     0     0     0     0     0     0
0       0.00000      00000    00TREF
   95     52     0     0     0     0     0     0     0     0     0
0       0.00000     .00000    00
  160     47    48    52    50     0     0     0     0     0     0
0    53.00000     .00000    00
   70     54    55     0     0     0     0     0     0     0     0
0       0.00000     .00000    00
   24     55    56    58     0     0     0     0     0     0     0
0       0.00000     .00000    00NXT
   33     56    57     0     0    24     0     0     0     0     0
0       0.00000     .00000    00
  157     57    23     0     0     0     0     0     0     0     0
0       0.00000     .00000    00LREF
   95     58     0     0     0     0     0     0     0     0     0
0       0.00000     .00000    00
  160     53    54    58    56     0     0     0     0     0     0
0       0.00000     .00000    00
  149     32    33     0    34     0     0     0     0     0     0
0    59.00000     .00000    00
  150     33    34     0     0     0     0     0     0     0     0
0       0.00000     .00000    00LEADER
  123     61    62     0     0     0     0     0     0     0     0
0       0.00000     .00000    00
   70     63    64     0     0     0     0     0     0     0     0
0       0.00000     .00000    00
   24     64    65     0     0     0     0     0     0     0     0
0       0.00000     .00000    00ID
   33     65    66     0     0    24     0     0     0     0     0
0       0.00000     .00000    00
  157     66    23     0     0     0     0     0     0     0     0
0       0.00000     .00000    00LREF
  160     62    63     0    65     0     0     0     0     0     0
0    67.00000     .00000    00
   70     68    69     0     0     0     0     0     0     0     0
0       0.00000     .00000    00
   24     69    70     0     0     0     0     0     0     0     0
0       0.00000     .00000    00NXT
   33     70    71     0     0    29     0     0     0     0     0
0       0.00000     .00000    00
```

```
  157    71    28     0     0     0     0     0     0     0     0
0     0.00000     .00000    00TREF
  160    67    68     0    70     0     0     0     0     0     0
0     0.00000     .00000    00
  149    59    60     0    61     0     0     0     0     0     0
0    77.00000     .00000    00
  150    60    61     0     0     0     0     0     0     0     0
0     0.00000     .00000    00TRAILER
   70    72    73     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  161    73    78     0     0     0     0     0     0     0     0
0    74.00000     .00000    00HEAD
  161    74    78     0     0     0     0     0     0     0     0
0    75.00000     .00000    00TAIL
  161    75    78     0     0     0     0     0     0     0     0
0    76.00000     .00000    00P
  161    76    78     0     0     0     0     0     0     0     0
0     0.00000     .00000    00Q
  160    77    72     0    78     0     0     0     0     0     0
0    82.00000     .00000    00
   33    78    79     0     0    24     0     0     0     0     0
0     0.00000     .00000    00
  157    79    23     0     0     0     0     0     0     0     0
0     0.00000     .00000    00LREF
   70    80    81     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  161    81    83     0     0     0     0     0     0     0     0
0     0.00000     .00000    00T
  160    82    80     0    83     0     0     0     0     0     0
0    87.00000     .00000    00
   33    83    84     0     0    29     0     0     0     0     0
0     0.00000     .00000    00
  157    84    28     0     0     0     0     0     0     0     0
0     0.00000     .00000    00TREF
   70    85    86     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  161    86    88     0     0     0     0     0     0     0     0
0     0.00000     .00000    00Z
  160    87    85     0    88     0     0     0     0     0     0
0    93.00000     .00000    00
   33    88    89     0     4     4     0     0     0     0     0
0     0.00000     .00000    00
  157    89     4     0     0     0     0     0     0     0     0
0     0.00000     .00000    00INTEGER
   70    90    91     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  161    91    94     0     0     0     0     0     0     0     0
0    92.00000     .00000    00X
  161    92    94     0     0     0     0     0     0     0     0
0     0.00000     .00000    00Y
  110    98    99     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
   72    99   100   102   104     0     0     0     0     0     0
0     0.00000     .00000    00
   70   100   101     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
   73   101   102   104     0     0     0     0     0     0     0
0     0.00000     .00000    00W
   33   102   103     0     4     4     0     0     0     0     0
```

```
0      0.00000      .00000     00
  157    103      4      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOINTEGER
  166    104      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
  160     93     90      0     94      0      0      0      0      0      0
0    107.00000      .00000     00
   33     94     95      0      4      4      0      0      0      0      0
0      0.00000      .00000     00
  157     95      4      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOINTEGER
   61     96     98    105      0      0      0      0      0      0      0
0      0.00000      .00000     00
   63     97     96    108      0      0      0      0      0      0      0
0      0.00000      .00000     OOL
   33    105    106      0      0     24      0      0      0      0      0
0      0.00000      .00000     00
  157    106     23      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOLREF
   82    109    112      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
   70    110    111      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
  161    111    113      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOH
  134    115    117      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    116    111      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOH
   13    117    116    118      0      0      0      0      0      0      0
0    122.00000      .00000     00
  157    118     73      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOHEAD
  157    119     74      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOTAIL
  131    120    119    121      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    121     37      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOKEY
   13    122    120    123      0      0      0      0      0      0      0
0    124.00000      .00000     00
  157    123    101      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOW
  134    133    135      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    134    111      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOH
   13    135    134    137      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    136    111      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOH
  131    137    136    138      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    138     55      0      0      0      0      0      0      0      0
0      0.00000      .00000     OONXT
   88    124    125    133      0      0      0      0      0      0      0
0    139.00000      .00000     00
  167    125    129      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
```

```
  157   126    74     0      0     0     0     0     0     0     0
0      0.00000      .00000     OOTAIL
  131   127   126   128      0     0     0     0     0     0     0
0    132.00000      .00000     OO
  157   128    37     0      0     0     0     0     0     0     0
0      0.00000      .00000     OOKEY
   62   129   130   131      0     0     0     0     0     0     0
0      0.00000      .00000     OO
  138   130     0     0      0     0     0     0     0     0     0
0      0.00000      .00000     OO/=
  109   131   127     0      0     0     0     0     0     0     0
0      0.00000      .00000     OO
  157   132   101     0      0     0     0     0     0     0     0
0      0.00000      .00000     OOW
  134   146   148     0      0     0     0     0     0     0     0
0      0.00000      .00000     OO
  157   147    74     0      0     0     0     0     0     0     0
0      0.00000      .00000     OOTAIL
   13   148   147   149      0     0     0     0     0     0     0
0    153.00000      .00000     OO
    7   149   150   151      0     0     0     0     0     0     0
0      0.00000      .00000     OO
  157   150    33     0      0     0     0     0     0     0     0
0      0.00000      .00000     OOLEADER
  166   151     0     0      0     0     0     0     0     0     0
0      0.00000      .00000     OO
  157   152    86     0      0     0     0     0     0     0     0
0      0.00000      .00000     OOZ
   62   155   156   157      0     0     0     0     0     0     0
0      0.00000      .00000     OO
  138   156     0     0      0     0     0     0     0     0     0
0      0.00000      .00000     OO+
  109   157   154     0      0     0     0     0     0     0     0
0      0.00000      .00000     OO
  100   158     4     0      0     0     0     0     0     0     0
0      0.10000+001  .00000     101
  157   159   111     0      0     0     0     0     0     0     0
0      0.00000      .00000     OOH
  131   160   159   161      0     0     0     0     0     0     0
0      0.00000      .00000     OO
  157   161    43     0      0     0     0     0     0     0     0
0      0.00000      .00000     OOCOUNT
   13   153   152   155      0     0     0     0     0     0     0
0    162.00000      .00000     OO
  157   154    86     0      0     0     0     0     0     0     0
0    158.00000      .00000     OOZ
   13   162   160   163      0     0     0     0     0     0     0
0    167.00000      .00000     OO
  100   163     4     0      0     0     0     0     0     0     0
0      0.00000      .00000     100
  157   164   111     0      0     0     0     0     0     0     0
0      0.00000      .00000     OOH
  131   165   164   166      0     0     0     0     0     0     0
0      0.00000      .00000     OO
  157   166    49     0      0     0     0     0     0     0     0
0      0.00000      .00000     OOTRAIL
   13   167   165   168      0     0     0     0     0     0     0
0    172.00000      .00000     OO
   95   168     0     0      0     0     0     0     0     0     0
```

```
0      0.00000      .00000     00
  157    169    111      0      0      0      0      0      0      0      0
0      0.00000      .00000     00H
  131    170    169    171      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    171     55      0      0      0      0      0      0      0      0
0      0.00000      .00000     00NXT
   13    172    170    173      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    173     74      0      0      0      0      0      0      0      0
0      0.00000      .00000     00TAIL
   71    139    140      0      0      0      0      0      0      0      0
0    174.00000      .00000     00
   29    140    142    146      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    141    111      0      0      0      0      0      0      0      0
0    145.00000      .00000     00H
   62    142    143    144      0      0      0      0      0      0      0
0      0.00000      .00000     00
  138    143      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     00=
  109    144    141      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    145     74      0      0      0      0      0      0      0      0
0      0.00000      .00000     00TAIL
  126    174    175      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    175    111      0      0      0      0      0      0      0      0
0      0.00000      .00000     00H
  160    112    110      0    113      0      0      0      0      0      0
0      0.00000      .00000     00
   33    113    114      0      0     24      0      0      0      0      0
0      0.00000      .00000     00
  157    114     23      0      0      0      0      0      0      0      0
0      0.00000      .00000     00LREF
   19    108    109    115      0      0      0      0      0      0      0
0      0.00000      .00000     00
  134    176    178      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    177     73      0      0      0      0      0      0      0      0
0      0.00000      .00000     00HEAD
   13    178    177    179      0      0      0      0      0      0      0
0    183.00000      .00000     00
    7    179    180    181      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    180     33      0      0      0      0      0      0      0      0
0      0.00000      .00000     00LEADER
  166    181      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    182     74      0      0      0      0      0      0      0      0
0      0.00000      .00000     00TAIL
   13    183    182    184      0      0      0      0      0      0      0
0    186.00000      .00000     00
  157    184     73      0      0      0      0      0      0      0      0
0      0.00000      .00000     00HEAD
  157    185     86      0      0      0      0      0      0      0      0
0      0.00000      .00000     00Z
   13    186    185    187      0      0      0      0      0      0      0
0    189.00000      .00000     00
```

```
  100   187     4     0     0     0     0     0     0     0     0
0      0.00000      .00000     100
  157   188     0     0     0     0     0     0     0     0     0
0      0.00000      .00000     00GET
  109   190   191     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  119   189   188   190     0     0     0     0     0     0     0
0    192.00000      .00000     00
  157   191    91     0     0     0     0     0     0     0     0
0      0.00000      .00000     00X
  134   199   201     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  157   200     0     0     0     0     0     0     0     0     0
0      0.00000      .00000     00GET
  109   202   203     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  119   201   200   202     0     0     0     0     0     0     0
0    205.00000      .00000     00
  157   203    92     0     0     0     0     0     0     0     0
0      0.00000      .00000     00Y
  157   204     0     0     0     0     0     0     0     0     0
0      0.00000      .00000     00PUT
  109   206   207     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  119   205   204   206     0     0     0     0     0     0     0
0    209.00000      .00000     00
  157   207    91     0     0     0     0     0     0     0     0
0      0.00000      .00000     00X
  157   208     0     0     0     0     0     0     0     0     0
0      0.00000      .00000     00PUT
  109   210   211     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  119   209   208   210     0     0     0     0     0     0     0
0    213.00000      .00000     00
  157   211    92     0     0     0     0     0     0     0     0
0      0.00000      .00000     00Y
  157   212    75     0     0     0     0     0     0     0     0
0      0.00000      .00000     00P
  109   216   217     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
   13   213   212   214     0     0     0     0     0     0     0
0    219.00000      .00000     00
   62   214   215   216     0     0     0     0     0     0     0
0      0.00000      .00000     00
  157   215    97     0     0     0     0     0     0     0     0
0      0.00000      .00000     00L
  157   217    91     0     0     0     0     0     0     0     0
0      0.00000      .00000     00X
  157   218    76     0     0     0     0     0     0     0     0
0      0.00000      .00000     00Q
  109   222   223     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
   13   219   218   220     0     0     0     0     0     0     0
0    225.00000      .00000     00
   62   220   221   222     0     0     0     0     0     0     0
0      0.00000      .00000     00
  157   221    97     0     0     0     0     0     0     0     0
0      0.00000      .00000     00L
  157   223    92     0     0     0     0     0     0     0     0
```

```
0      0.00000      .00000     00Y
  157    224     81      0      0      0      0      0      0      0      0
0      0.00000      .00000     00T
   22    230    231      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
  109    238    235      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
   95    239      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    227     60      0      0      0      0      0      0      0      0
0      0.00000      .00000     00TRAILER
    5    228    229      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
   91    229    230    232      0      0      0      0      0      0      0
0    236.00000      .00000     00
  157    231     64      0      0      0      0      0      0      0      0
0      0.00000      .00000     00ID
   95    232      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    235     69      0      0      0      0      0      0      0      0
0    239.00000      .00000     00NXT
   62    236    237    238      0      0      0      0      0      0      0
0      0.00000      .00000     00
  138    237      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     00=
  157    240     81      0      0      0      0      0      0      0      0
0      0.00000      .00000     00T
  131    241    240    242      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    242     64      0      0      0      0      0      0      0      0
0      0.00000      .00000     00ID
   13    225    224    226      0      0      0      0      0      0      0
0    243.00000      .00000     00
    7    226    227    228      0      0      0      0      0      0      0
0      0.00000      .00000     00
   13    243    241    244      0      0      0      0      0      0      0
0    248.00000      .00000     00
  157    244     76      0      0      0      0      0      0      0      0
0      0.00000      .00000     00Q
  157    245     81      0      0      0      0      0      0      0      0
0      0.00000      .00000     00T
  131    246    245    247      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    247     69      0      0      0      0      0      0      0      0
0      0.00000      .00000     00NXT
   13    248    246    250      0      0      0      0      0      0      0
0    255.00000      .00000     00
  157    249     75      0      0      0      0      0      0      0      0
0      0.00000      .00000     00P
  131    250    249    251      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    251     49      0      0      0      0      0      0      0      0
0      0.00000      .00000     00TRAIL
  157    252     75      0      0      0      0      0      0      0      0
0      0.00000      .00000     00P
  131    253    252    254      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    254     49      0      0      0      0      0      0      0      0
0      0.00000      .00000     00TRAIL
```

```
   13   255   253   256     0     0     0     0     0     0     0
0   260.00000     .00000    00
  157   256    81     0     0     0     0     0     0     0     0
0     0.00000     .00000    00T
  157   257    76     0     0     0     0     0     0     0     0
0     0.00000     .00000    00Q
  131   258   257   259     0     0     0     0     0     0     0
0     0.00000     .00000    00
  157   259    43     0     0     0     0     0     0     0     0
0     0.00000     .00000    00COUNT
   13   260   258   264     0     0     0     0     0     0     0
0   269.00000     .00000    00
  157   261    76     0     0     0     0     0     0     0     0
0     0.00000     .00000    00Q
  131   262   261   263     0     0     0     0     0     0     0
0   267.00000     .00000    00
  157   263    43     0     0     0     0     0     0     0     0
0     0.00000     .00000    00COUNT
   62   264   265   266     0     0     0     0     0     0     0
0     0.00000     .00000    00
  138   265     0     0     0     0     0     0     0     0     0
0     0.00000     .00000    00+
  109   266   262     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  100   267     4     0     0     0     0     0     0     0     0
0     0.10000+001 .00000    101
  157   268     0     0     0     0     0     0     0     0     0
0     0.00000     .00000    00GET
  109   270   271     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  119   269   268   270     0     0     0     0     0     0     0
0     0.00000     .00000    00
  157   271    91     0     0     0     0     0     0     0     0
0     0.00000     .00000    00X
  157   272    75     0     0     0     0     0     0     0     0
0     0.00000     .00000    00P
   88   192   193   199     0     0     0     0     0     0     0
0   273.00000     .00000    00
  167   193   195     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  157   194    91     0     0     0     0     0     0     0     0
0   198.00000     .00000    00X
   62   195   196   197     0     0     0     0     0     0     0
0     0.00000     .00000    00
  138   196     0     0     0     0     0     0     0     0     0
0     0.00000     .00000    00/=
  109   197   194     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  100   198     4     0     0     0     0     0     0     0     0
0     0.00000     .00000    100
   13   273   272   274     0     0     0     0     0     0     0
0   276.00000     .00000    00
  157   274    73     0     0     0     0     0     0     0     0
0     0.00000     .00000    00HEAD
  157   275    73     0     0     0     0     0     0     0     0
0     0.00000     .00000    00HEAD
   13   276   275   277     0     0     0     0     0     0     0
0   278.00000     .00000    00
   95   277     0     0     0     0     0     0     0     0     0
```

```
0      0.00000      .00000     00
  134    285    287      0       0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    286     76      0       0      0      0      0      0      0      0
0      0.00000      .00000     00Q
   13    287    286    288       0      0      0      0      0      0      0
0    290.00000      .00000     00
  157    288     75      0       0      0      0      0      0      0      0
0      0.00000      .00000     00P
  157    289     75      0       0      0      0      0      0      0      0
0      0.00000      .00000     00P
   13    290    289    292       0      0      0      0      0      0      0
0    294.00000      .00000     00
  157    291     75      0       0      0      0      0      0      0      0
0      0.00000      .00000     00P
  131    292    291    293       0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    293     55      0       0      0      0      0      0      0      0
0      0.00000      .00000     00NXT
  134    303    307      0       0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    304     76      0       0      0      0      0      0      0      0
0      0.00000      .00000     00Q
  131    305    304    306       0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    306     55      0       0      0      0      0      0      0      0
0      0.00000      .00000     00NXT
   13    307    305    308       0      0      0      0      0      0      0
0    310.00000      .00000     00
  157    308     73      0       0      0      0      0      0      0      0
0      0.00000      .00000     00HEAD
  157    309     73      0       0      0      0      0      0      0      0
0      0.00000      .00000     00HEAD
   13    310    309    311       0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    311     76      0       0      0      0      0      0      0      0
0      0.00000      .00000     00Q
   71    294    295      0       0      0      0      0      0      0      0
0      0.00000      .00000     00
   29    295    299    303       0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    296     76      0       0      0      0      0      0      0      0
0      0.00000      .00000     00Q
  131    297    296    298       0      0      0      0      0      0      0
0    302.00000      .00000     00
  157    298     43      0       0      0      0      0      0      0      0
0      0.00000      .00000     00COUNT
   62    299    300    301       0      0      0      0      0      0      0
0      0.00000      .00000     00
  138    300      0      0       0      0      0      0      0      0      0
0      0.00000      .00000     00=
  109    301    297      0       0      0      0      0      0      0      0
0      0.00000      .00000     00
  100    302      4      0       0      0      0      0      0      0      0
0      0.00000      .00000     100
  157    312     76      0       0      0      0      0      0      0      0
0      0.00000      .00000     00Q
   88    278    279    285       0      0      0      0      0      0      0
0    313.00000      .00000     00
```

```
  167   279   281     0      0      0     0     0     0     0     0
0      0.00000      .00000     00
  157   280    75     0      0      0     0     0     0     0     0
0    284.00000      .00000     00P
   62   281   282   283      0      0     0     0     0     0     0
0      0.00000      .00000     00
  138   282     0     0      0      0     0     0     0     0     0
0      0.00000      .00000     00/=
  109   283   280     0      0      0     0     0     0     0     0
0      0.00000      .00000     .00
  157   284    74     0      0      0     0     0     0     0     0
0      0.00000      .00000     00TAIL
   13   313   312   314      0      0     0     0     0     0     0
0    316.00000      .00000     00
  157   314    73     0      0      0     0     0     0     0     0
0      0.00000      .00000     00HEAD
  134   324   326     0      0      0     0     0     0     0     0
0      0.00000      .00000     00
  157   325     0     0      0      0     0     0     0     0     0
0      0.00000      .00000     00PUT
  109   327   329     0      0      0     0     0     0     0     0
0      0.00000       00000     00
  119   326   325   327      0      0     0     0     0     0     0
0    332.00000      .00000     00
  157   328    76     0      0      0     0     0     0     0     0
0      0.00000      .00000     00Q
  131   329   328   330      0      0     0     0     0     0     0
0      0.00000      .00000     00
  157   330    37     0      0      0     0     0     0     0     0
0      0.00000      .00000     00KEY
  157   331    86     0      0      0     0     0     0     0     0
0      0.00000      .00000     00Z
   13   332   331   334      0      0     0     0     0     0     0
0    339.00000      .00000     00
  157   333    86     0      0      0     0     0     0     0     0
0    337.00000      .00000     00Z
   62   334   335   336      0      0     0     0     0     0     0
0      0.00000      .00000     00
  138   335     0     0      0      0     0     0     0     0     0
0      0.00000      .00000     00-
  109   336   333     0      0      0     0     0     0     0     0
0      0.00000      .00000     00
  100   337     4     0      0      0     0     0     0     0     0
0      0.10000+001.00000     101
  157   338    81     0      0      0     0     0     0     0     0
0      0.00000      .00000     00T
   13   339   338   341      0      0     0     0     0     0     0
0    344.00000      .00000     00
  157   340    76     0      0      0     0     0     0     0     0
0      0.00000      .00000     00Q
  131   341   340   342      0      0     0     0     0     0     0
0      0.00000      .00000     00
  157   342    49     0      0      0     0     0     0     0     0
0      0.00000      .00000     00TRAIL
  157   343    76     0      0      0     0     0     0     0     0
0      0.00000      .00000     00Q
   13   344   343   346      0      0     0     0     0     0     0
0    349.00000      .00000     00
  157   345    76     0      0      0     0     0     0     0     0
```

```
0      0.00000      .00000     00Q
  131    346    345    347      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    347     55      0      0      0      0      0      0      0      0
0      0.00000      .00000     00NXT
  134    357    359      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    358     75      0      0      0      0      0      0      0      0
0      0.00000      .00000     00P
   13    359    358    361      0      0      0      0      0      0      0
0    366.00000      .00000     00
  157    360     81      0      0      0      0      0      0      0      0
0      0.00000      .00000     00T
  131    361    360    362      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    362     64      0      0      0      0      0      0      0      0
0      0.00000      .00000     00ID
  157    363     75      0      0      0      0      0      0      0      0
0      0.00000      .00000     00P
  131    364    363    365      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    365     43      0      0      0      0      0      0      0      0
0      0.00000      .00000     00COUNT
   62    370    371    372      0      0      0      0      0      0      0
0      0.00000      .00000     00
  138    371      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     00-
  109    372    368      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
  100    373      4      0      0      0      0      0      0      0      0
0      0.10000+001.00000     101
   13    366    364    370      0      0      0      0      0      0      0
0    374.00000      .00000     00
  157    367     75      0      0      0      0      0      0      0      0
0      0.00000      .00000     00P
  131    368    367    369      0      0      0      0      0      0      0
0    373.00000      .00000     00
  157    369     43      0      0      0      0      0      0      0      0
0      0.00000      .00000     00COUNT
  134    383    387      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    384     75      0      0      0      0      0      0      0      0
0      0.00000      .00000     00P
  131    385    384    386      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    386     55      0      0      0      0      0      0      0      0
0      0.00000      .00000     00NXT
   13    387    385    388      0      0      0      0      0      0      0
0    390.00000      .00000     00
  157    388     76      0      0      0      0      0      0      0      0
0      0.00000      .00000     00Q
  157    389     76      0      0      0      0      0      0      0      0
0      0.00000      .00000     00Q
   13    390    389    391      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    391     75      0      0      0      0      0      0      0      0
0      0.00000      .00000     00P
   71    374    375      0      0      0      0      0      0      0      0
0    393.00000      .00000     00
```

```
   29   375   379   383     0     0     0     0     0     0     0
0      0.00000      .00000    00
  157   376    75     0      0     0     0     0     0     0     0
0      0.00000      .00000    00P
  131   377   376   378     0     0     0     0     0     0     0
0    382.00000      .00000    00
  157   378    43     0      0     0     0     0     0     0     0
0      0.00000      .00000    00COUNT
   62   379   380   381     0     0     0     0     0     0     0
0      0.00000      .00000    00
  138   380     0     0      0     0     0     0     0     0     0
0      0.00000      .00000    00=
  109   381   377     0      0     0     0     0     0     0     0
0      0.00000      .00000    00
  100   382     4     0      0     0     0     0     0     0     0
0      0.00000      .00000    100
  157   392    81     0      0     0     0     0     0     0     0
0      0.00000      .00000    00T
   13   393   392   395     0     0     0     0     0     0     0
0      0.00000      .00000    00
  157   394    81     0      0     0     0     0     0     0     0
0      0.00000      .00000    00T
  131   395   394   396     0     0     0     0     0     0     0
0      0.00000      .00000    00
  157   396    69     0      0     0     0     0     0     0     0
0      0.00000      .00000    00NXT
   86   348   350     0      0     0     0     0     0     0     0
0      0.00000      .00000    00INNER
   92   349   348   350     0     0     0     0     0     0     0
0      0.00000      .00000    00
   88   350   351   357     0     0     0     0     0     0     0
0      0.00000      .00000    00
  167   351   353     0      0     0     0     0     0     0     0
0      0.00000      .00000    00
  157   352    81     0      0     0     0     0     0     0     0
0    356.00000      .00000    00T
   62   353   354   355     0     0     0     0     0     0     0
0      0.00000      .00000    00
  138   354     0     0      0     0     0     0     0     0     0
0      0.00000      .00000    00/=
  109   355   352     0      0     0     0     0     0     0     0
0      0.00000      .00000    00
   95   356     0     0      0     0     0     0     0     0     0
0      0.00000      .00000    00
   92   316   315   317     0     0     0     0     0     0     0
0    397.00000      .00000    00
   88   317   318   324     0     0     0     0     0     0     0
0      0.00000      .00000    00
  167   318   320     0      0     0     0     0     0     0     0
0      0.00000      .00000    00
  157   319    76     0      0     0     0     0     0     0     0
0    323.00000      .00000    00Q
   62   320   321   322     0     0     0     0     0     0     0
0      0.00000      .00000    00
  138   321     0     0      0     0     0     0     0     0     0
0      0.00000      .00000    00/=
  109   322   319     0      0     0     0     0     0     0     0
0      0.00000      .00000    00
   95   323     0     0      0     0     0     0     0     0     0
```

```
0       0.00000      .00000     00
  134     404    406      0      0      0      0      0      0      0      0
0       0.00000      .00000     00
  157     405      0      0      0      0      0      0      0      0      0
0       0.00000      .00000     00PUT
  109     407    408      0      0      0      0      0      0      0      0
0       0.00000      .00000     00
  119     406    405    407      0      0      0      0      0      0      0
0       0.00000      .00000     00
  135     408      0      0      0      0      0      0      0      0      0
409     0.00000      .00000     00"THIS SET IS NOT
  487     409      0      0      0      0      0      0      0      0      0
410     0.00000      .00000     00 PARTIALLY ORDER
  487     410      0      0      0      0      0      0      0      0      0
0       0.00000      .00000     00ED."
  157     399     86      0      0      0      0      0      0      0      0
0   403.00000      .00000     00Z
   62     400    401    402      0      0      0      0      0      0      0
0       0.00000      .00000     00
  138     401      0      0      0      0      0      0      0      0      0
0       0.00000      .00000     00=
  109     402    399      0      0      0      0      0      0      0      0
0       0.00000      .00000     00
  100     403      4      0      0      0      0      0      0      0      0
0       0.00000      .00000     100
   86     315    317      0      0      0      0      0      0      0      0
0       0.00000      .00000     000UTER
   71     397    398      0      0      0      0      0      0      0      0
0       0.00000      .00000     00
   29     398    400    404      0      0      0      0      0      0      0
0       0.00000      .00000     00
   19      16     17    176      0      0      0      0      0      0      0
0       0.00000      .00000     00
  137     107     97     96    108      0      0      0      0      0      0
0       0.00000      .00000     00
   27      12      0      0     15      0      0      0      0      0      0
0       0.00000      .00000     00
```

12.2. APPENDIX IVb.

```
   28    11    12     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  118    13     0     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  117    14    13    16     0     0     0     0     0     0     0
0      0.00000      .00000     00INSERT
  137    15    14    13    16     0     0     0     0     0     0
0      0.00000      .00000     00
   82    17    18     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  149    18    19     0     0     0     0     0     0     0     0
0     20.00000      .00000     00
  150    19     0     0     0     0     0     0     0     0     0
0      0.00000      .00000     00NODE
  149    20    21     0    22     0     0     0     0     0     0
0     25.00000      .00000     00
  150    21    22     0     0     0     0     0     0     0     0
0      0.00000      .00000     00REF
    3    22    23     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
   33    23    24     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  157    24    19     0     0     0     0     0     0     0     0
0      0.00000      .00000     00NODE
  123    27    28     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
   70    29    30     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
   24    30    31     0     0     0     0     0     0     0     0
0      0.00000      .00000     00KEY
   33    31    32     0     4     4     0     0     0     0     0
0      0.00000      .00000     00
  157    32     4     0     0     0     0     0     0     0     0
0      0.00000      .00000     00INTEGER
  160    28    29     0    31     0     0     0     0     0     0
0     33.00000      .00000     00
   70    34    35     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
   24    35    36     0     0     0     0     0     0     0     0
0      0.00000      .00000     00COUNT
   33    36    37     0     4     4     0     0     0     0     0
0      0.00000      .00000     00
  157    37     4     0     0     0     0     0     0     0     0
0      0.00000      .00000     00INTEGER
  160    33    34     0    36     0     0     0     0     0     0
0     38.00000      .00000     00
   70    39    40     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
   24    40    41     0     0     0     0     0     0     0     0
0      0.00000      .00000     00NXT
   33    41    42     0     0    22     0     0     0     0     0
0      0.00000      .00000     00
  157    42    21     0     0     0     0     0     0     0     0
0      0.00000      .00000     00REF
  160    38    39     0    41     0     0     0     0     0     0
0      0.00000      .00000     00
```

```
  149    25    26     0    27     0     0     0     0     0     0
0    45.00000     .00000    00
  150    26    27     0     0     0     0     0     0     0     0
0     0.00000     .00000    00NODE
   70    43    44     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  161    44    46     0     0     0     0     0     0     0     0
0     0.00000     .00000    00K
  160    45    43     0    46     0     0     0     0     0     0
0    50.00000     .00000    00
   33    46    47     0     4     4     0     0     0     0     0
0     0.00000     .00000    00
  157    47     4     0     0     0     0     0     0     0     0
0     0.00000     .00000    00INTEGER
  100    61     4     0     0     0     0     0     0     0     0
0    64.00000     .00000    100
   95    64     0     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  157    54    26     0     0     0     0     0     0     0     0
0     0.00000     .00000    00NODE
    5    55    58     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  100    58     4     0     0     0     0     0     0     0     0
0    61.00000     .00000    100
   70    48    49     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  161    49    51    53     0     0     0     0     0     0     0
0     0.00000     .00000    00ROOT
  110    67    68     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
   72    68    69    71    73     0     0     0     0     0     0
0    74.00000     .00000    00
   70    69    70     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
   73    70    71    73     0     0     0     0     0     0     0
0     0.00000     .00000    00X
   33    71    72     0     4     4     0     0     0     0     0
0     0.00000     .00000    00
  157    72     4     0     0     0     0     0     0     0     0
0     0.00000     .00000    00INTEGER
  166    73     0     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
   75    74    75    77    79     0     0     0     0     0     0
0     0.00000     .00000    00
   70    75    76     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
   76    76    77     0     0     0     0     0     0     0     0
0     0.00000     .00000    00BASE
   33    77    78     0     0    22     0     0     0     0     0
0     0.00000     .00000    00
  157    78    21     0     0     0     0     0     0     0     0
0     0.00000     .00000    00REF
  166    79     0     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  160    50    48    53    51     0     0     0     0     0     0
0    80.00000     .00000    00
  118    65    67     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  117    66    65    81     0     0     0     0     0     0     0
```

```
0      0.00000      .00000     00SEARCH
    33     51     52      0      0     22      0      0      0      0      0
0      0.00000      .00000     00
   157     52     21      0      0      0      0      0      0      0      0
0      0.00000      .00000     00REF
     7     53     54     55      0      0      0      0      0      0      0
0      0.00000      .00000     00
    82     82     85      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
    70     83     84      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
   161     84     86      0      0      0      0      0      0      0      0
0      0.00000      .00000     00W
   160     85     83      0     86      0      0      0      0      0      0
0     90.00000      .00000     00
    33     86     87      0      0     22      0      0      0      0      0
0      0.00000      .00000     00
   157     87     21      0      0      0      0      0      0      0      0
0      0.00000      .00000     00REF
    70     88     89      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
   161     89     91      0      0      0      0      0      0      0      0
0      0.00000      .00000     00B
   134     93     95      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
   157     94     84      0      0      0      0      0      0      0      0
0      0.00000      .00000     00W
    13     95     94     96      0      0      0      0      0      0      0
0     98.00000      .00000     00
   157     96     76      0      0      0      0      0      0      0      0
0      0.00000      .00000     00BASE
   157     97     89      0      0      0      0      0      0      0      0
0      0.00000      .00000     00B
    13     98     97     99      0      0      0      0      0      0      0
0    100.00000      .00000     00
   158     99      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     10TRUE
   134    111    112      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
   134    121    123      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
   157    122     89      0      0      0      0      0      0      0      0
0      0.00000      .00000     00B
    13    123    122    124      0      0      0      0      0      0      0
0      0.00000      .00000     00
   158    124      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     10FALSE
    29    113    117    121      0      0      0      0      0      0      0
0    125.00000      .00000     00
    29    125    126    127      0      0      0      0      0      0      0
0      0.00000      .00000     00
   134    127    129      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
   157    128     84      0      0      0      0      0      0      0      0
0      0.00000      .00000     00W
    13    129    128    131      0      0      0      0      0      0      0
0      0.00000      .00000     00
   157    130     84      0      0      0      0      0      0      0      0
0      0.00000      .00000     00W
```

```
  131   131   130   132     0     0     0     0     0     0     0
0      0.00000     .00000    00
  157   132    40     0     0     0     0     0     0     0     0
0      0.00000     .00000    00NXT
   71   112   113     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
  157   114    84     0     0     0     0     0     0     0     0
0      0.00000     .00000    00W
  131   115   114   116     0     0     0     0     0     0     0
0    120.00000     .00000    00
  157   116    30     0     0     0     0     0     0     0     0
0      0.00000     .00000    00KEY
   62   117   118   119     0     0     0     0     0     0     0
0      0.00000     .00000    00
  138   118     0     0     0     0     0     0     0     0     0
0      0.00000     .00000    00=
  109   119   115     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
  157   120    70     0     0     0     0     0     0     0     0
0      0.00000     .00000    00X
  166   126     0     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
   62   107   108   109     0     0     0     0     0     0     0
0      0.00000     .00000    00
  138   108     0     0     0     0     0     0     0     0     0
0      0.00000     .00000    00AND
  109   109   103     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
  157   110    89     0     0     0     0     0     0     0     0
0      0.00000     .00000    00B
   88   100   101   111     0     0     0     0     0     0     0
0    133.00000     .00000    00
  167   101   107     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
  157   102    84     0     0     0     0     0     0     0     0
0    106.00000     .00000    00W
   62   103   104   105     0     0     0     0     0     0     0
0    110.00000     .00000    00
  138   104     0     0     0     0     0     0     0     0     0
0      0.00000     .00000    00/=
  109   105   102     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
   95   106     0     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
  134   136   138     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
  157   137    84     0     0     0     0     0     0     0     0
0      0.00000     .00000    00W
   13   138   137   139     0     0     0     0     0     0     0
0    141.00000     .00000    00
  157   139    76     0     0     0     0     0     0     0     0
0      0.00000     .00000    00BASE
  157   140    76     0     0     0     0     0     0     0     0
0      0.00000     .00000    00BASE
  100   150     4     0     0     0     0     0     0     0     0
0    153.00000     .00000    100
   95   153     0     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
  157   143    26     0     0     0     0     0     0     0     0
```

```
0      0.00000      .00000     OONODE
     5    144    147      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO
   100    147      4      0      0      0      0      0      0      0      0
0    150.00000      .00000     100
   157    154     76      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOBASE
   131    155    154    156      0      0      0      0      0      0      0
0      0.00000      .00000     OO
   157    156     30      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOKEY
    13    141    140    142      0      0      0      0      0      0      0
0    157.00000      .00000     OO
     7    142    143    144      0      0      0      0      0      0      0
0      0.00000      .00000     OO
    13    157    155    158      0      0      0      0      0      0      0
0    162.00000      .00000     OO
   157    158     70      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOX
   157    159     76      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOBASE
   131    160    159    161      0      0      0      0      0      0      0
0      0.00000      .00000     OO
   157    161     35      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOCOUNT
    13    162    160    163      0      0      0      0      0      0      0
0      0.00000      00000      OO
   100    163      4      0      0      0      0      0      0      0      0
0      0.10000+001.00000      101
    29    134    135    136      0      0      0      0      0      0      0
0    164.00000      .00000     OO
    29    164    165    166      0      0      0      0      0      0      0
0      0.00000      .00000     OO
   134    166    170      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO
   157    167     84      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOW
   131    168    167    169      0      0      0      0      0      0      0
0      0.00000      .00000     OO
   157    169     35      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOCOUNT
    13    170    168    174      0      0      0      0      0      0      0
0      0.00000      .00000     OO
   157    171     84      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOW
   131    172    171    173      0      0      0      0      0      0      0
0    177.00000      .00000     OO
   157    173     35      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOCOUNT
    62    174    175    176      0      0      0      0      0      0      0
0      0.00000      .00000     OO
   138    175      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO+
   109    176    172      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO
   100    177      4      0      0      0      0      0      0      0      0
0      0.10000+001.00000      101
   166    165      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO
```

```
   71   133   134     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  157   135    89     0     0     0     0     0     0     0     0
0      0.00000      .00000     00B
   19    81    82    93     0     0     0     0     0     0     0
0      0.00000      .00000     00
  160    90    88     0    91     0     0     0     0     0     0
0      0.00000      .00000     00
   33    91    92     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  157    92     0     0     0     0     0     0     0     0     0
0      0.00000      .00000     00BOOLEAN
  110   180   181     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
   72   181   182   184   186     0     0     0     0     0     0
0      0.00000      .00000     00
   70   182   183     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
   73   183   184   186     0     0     0     0     0     0     0
0      0.00000      .00000     00V
   33   184   185     0     0    22     0     0     0     0     0
0      0.00000      .00000     00
  157   185    21     0     0     0     0     0     0     0     0
0      0.00000      .00000     00REF
  166   186     0     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  137    80    66    65    81     0     0     0     0     0     0
0    187.00000      .00000     00
  118   178   180     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  117   179   178   188     0     0     0     0     0     0     0
0      0.00000      .00000     00PRINTLIST
  134   189   190     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  134   197   199     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  157   198     0     0     0     0     0     0     0     0     0
0      0.00000      .00000     00PUT
  109   200   202     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  119   199   198   200     0     0     0     0     0     0     0
0    205.00000      .00000     00
  157   201   183     0     0     0     0     0     0     0     0
0      0.00000      .00000     00V
  131   202   201   203     0     0     0     0     0     0     0
0      0.00000      .00000     00
  157   203    30     0     0     0     0     0     0     0     0
0      0.00000      .00000     00KEY
  157   204     0     0     0     0     0     0     0     0     0
0      0.00000      .00000     00PUT
  109   206   208     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  119   205   204   206     0     0     0     0     0     0     0
0    211.00000      .00000     00
  157   207   183     0     0     0     0     0     0     0     0
0      0.00000      .00000     00V
  131   208   207   209     0     0     0     0     0     0     0
0      0.00000      .00000     00
  157   209    35     0     0     0     0     0     0     0     0
```

```
0      0.00000      .00000     OOCOUNT
  157    210    183      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOV
   13    211    210    213      0      0      0      0      0      0      0
0      0.00000      .00000     OO
  157    212    183      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOV
  131    213    212    214      0      0      0      0      0      0      0
0      0.00000      .00000     OO
  157    214     40      0      0      0      0      0      0      0      0
0      0.00000      .00000     OONXT
   88    190    191    197      0      0      0      0      0      0      0
0      0.00000      .00000     OO
  167    191    193      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO
  157    192    183      0      0      0      0      0      0      0      0
0    196.00000      .00000     OOV
   62    193    194    195      0      0      0      0      0      0      0
0      0.00000      .00000     OO
  138    194      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO/=
  109    195    192      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO
   95    196      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO
   19    188      0    189      0      0      0      0      0      0      0
0      0.00000      .00000     OO
  134    215    217      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO
  157    216      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOGET
  109    218    219      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO
  119    217    216    218      0      0      0      0      0      0      0
0    220.00000      .00000     OO
  157    219     44      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOK
  134    227    229      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO
  157    228     66      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOSEARCH
  109    230    231      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO
  157    231     44      0      0      0      0      0      0      0      0
0    232.00000      .00000     OOK
  119    229    228    230      0      0      0      0      0      0      0
0    234.00000      .00000     OO
  157    232     49      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOROOT
  157    233      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOGET
  109    235    236      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO
  157    236     44      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOK
  119    234    233    235      0      0      0      0      0      0      0
0      0.00000      .00000     OO
  157    237    179      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOPRINTLIST
```

```
   88   220   221   227     0     0     0     0     0     0     0
0   238.00000     .00000    00
  167   221   223     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  157   222    44     0     0     0     0     0     0     0     0
0   226.00000     .00000    00K
   62   223   224   225     0     0     0     0     0     0     0
0     0.00000     .00000    00
  138   224     0     0     0     0     0     0     0     0     0
0     0.00000     .00000    00/=
  109   225   222     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  100   226     4     0     0     0     0     0     0     0     0
0     0.00000     .00000    100
  109   239   240     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  119   238   237   239     0     0     0     0     0     0     0
0     0.00000     .00000    00
  157   240    49     0     0     0     0     0     0     0     0
0     0.00000     .00000    00ROOT
   19    16    17   215     0     0     0     0     0     0     0
0     0.00000     .00000    00
  137   187   179   178   188     0     0     0     0     0     0
0     0.00000     .00000    00
   27    12     0     0    15     0     0     0     0     0     0
0     0.00000     .00000    00
```

12.3. APPENDIX IVc.

```
   28    11    12     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  118    13     0     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  117    14    13    16     0     0     0     0     0     0     0
0     0.00000     .00000    00V
  137    15    14    13    16     0     0     0     0     0     0
0     0.00000     .00000    00
   82    17    18     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  149    18    19     0    20     0     0     0     0     0     0
0    24.00000     .00000    00
  150    19    20     0     0     0     0     0     0     0     0
0     0.00000     .00000    00INTEGER
   81    20    21     4     0     0     0     0     0     0     0
0     0.00000     .00000    00
  122    21    22    23     4     0     0     0     0     0     0
0     0.00000     .00000    00
  100    22     4     0     0     0     0     0     0     0     0
0     0.10000+001 .00000    101
  100    23     4     0     0     0     0     0     0     0     0
0     0.50000+002 .00000    1050
  149    24    25     0    26     0     0     0     0     0     0
0    33.00000     .00000    00
  150    25    26     0     0     0     0     0     0     0     0
0     0.00000     .00000    00COLOUR
   49    26    27     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
   48    27    26     0     0     0     0     0     0     0     0
0    28.00000     .00000    10RED
   48    28    26     0     0     0     0     0     0     0     0
0    29.10000+001 .00000    100RANGE
   48    29    26     0     0     0     0     0     0     0     0
0    30.20000+001 .00000    10YELLOW
   48    30    26     0     0     0     0     0     0     0     0
0     0.30000+001 .00000    10GREEN
   70    31    32     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  161    32    34     0     0     0     0     0     0     0     0
0     0.00000     .00000    00ROBOT
  160    33    31     0    34     0     0     0     0     0     0
0    39.00000     .00000    00
   33    34    35     0    26    26    26     0     0     0     0
0     0.00000     .00000    00
  157    35    25     0     0     0     0     0     0     0     0
0     0.00000     .00000    00COLOUR
   70    36    37     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
  161    37    40     0     0     0     0     0     0     0     0
0    38.00000     .00000    00VAR2
  161    38    40     0     0     0     0     0     0     0     0
0     0.00000     .00000    00VAR1
  134    42    43     0     0     0     0     0     0     0     0
0     0.00000     .00000    00
   87    43    44    46     0     0     0     0     0     0     0
0    48.00000     .00000    00
```

```
   86    44    46     0      0      0     0     0     0     0     0
0      0.00000      .00000     OOLAB1
  157    45    38     0      0      0     0     0     0     0     0
0      0.00000      .00000     OOVAR1
   13    46    45    47      0      0     0     0     0     0     0
0      0.00000      .00000     OO
  100    47     4     0      0      0     0     0     0     0     0
0      0.20000+001  .00000     102
   87    48    49    52      0      0     0     0     0     0     0
0     55.00000      .00000     OO
   86    49    52     0      0      0     0     0     0     0     0
0     50.00000      .00000     OOLAB2
   86    50    52     0      0      0     0     0     0     0     0
0      0.00000      .00000     OOLAB3
  157    51    37     0      0      0     0     0     0     0     0
0      0.00000      .00000     OOVAR2
   13    52    51    53      0      0     0     0     0     0     0
0      0.00000      .00000     OO
  157    53    38     0      0      0     0     0     0     0     0
0      0.00000      .00000     OOVAR1
  157    54    37     0      0      0     0     0     0     0     0
0      0.00000      .00000     OOVAR2
   13    55    54    56      0      0     0     0     0     0     0
0     57.00000      .00000     OO
  100    56     4     0      0      0     0     0     0     0     0
0      0.10000+002  .00000     1010
   97    57     0     0      0      0     0     0     0     0     0
0     58.00000      .00000     OO
   69    58    59     0      0      0     0     0     0     0     0
0     60.00000      .00000     OO
  157    59    49     0      0      0     0     0     0     0     0
0      0.00000      .00000     OOLAB2
  157    61    44     0      0      0     0     0     0     0     0
0      0.00000      .00000     OOLAB1
   69    60    61     0      0      0     0     0     0     0     0
0     62.00000      .00000     OO
   69    62    63     0      0      0     0     0     0     0     0
0     64.00000      .00000     OO
  157    63    50     0      0      0     0     0     0     0     0
0      0.00000      .00000     OOLAB3
  134    70    71     0      0      0     0     0     0     0     0
0      0.00000      .00000     OO
    9    72    74     0      0      0     0     0     0     0     0
0      0.00000      .00000     OO
  157    73    38     0      0      0     0     0     0     0     0
0      0.00000      .00000     OOVAR1
   22    75    76     0      0      0     0     0     0     0     0
0      0.00000      .00000     OO
  134    78    79     0      0      0     0     0     0     0     0
0      0.00000      .00000     OO
   82    80    83     0      0      0     0     0     0     0     0
0      0.00000      .00000     OO
   70    81    82     0      0      0     0     0     0     0     0
0      0.00000      .00000     OO
  161    82    84     0      0      0     0     0     0     0     0
0      0.00000      .00000     OOVAR4
  134    86    88     0      0      0     0     0     0     0     0
0      0.00000      .00000     OO
  157    87    82     0      0      0     0     0     0     0     0
```

```
0      0.00000      .00000     00VAR4
   13     88     87     90      0      0      0      0      0      0      0
0     95.00000      .00000     00
  100     89      3      0      0      0      0      0      0      0      0
0     93.45000+001.00000     104.5
   62     90     91     92      0      0      0      0      0      0      0
0      0.00000      .00000     00
  138     91      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     00*
  109     92     89      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
  100     93      3      0      0      0      0      0      0      0      0
0      0.33000+001.00000     103.3
  157     94     82      0      0      0      0      0      0      0      0
0      0.00000      .00000     00VAR4
   13     95     94     97      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157     96     82      0      0      0      0      0      0      0      0
0    100.00000      .00000     00VAR4
   62     97     98     99      0      0      0      0      0      0      0
0      0.00000      .00000     00
  138     98      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     00**
  109     99     96      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
  100    100      4      0      0      0      0      0      0      0      0
0      0.20000+001.00000     102
   19     79     80     86      0      0      0      0      0      0      0
0      0.00000      .00000     00
  160     83     81      0     84      0      0      0      0      0      0
0      0.00000      .00000     00
   33     84     85      0      3      3      0      0      0      0      0
0      0.00000      .00000     00
  157     85      3      0      0      0      0      0      0      0      0
0      0.00000      .00000     00FLOAT
    8     74     75     78      0      0      0      0      0      0      0
0    101.00000      .00000     00
  100     76      4      0      0      0      0      0      0      0      0
0      0.10000+001.00000     101
  100     77      4      0      0      0      0      0      0      0      0
0      0.20000+001.00000     102
   22    102    103      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
  134    104    105      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
  134    107    109      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    108     38      0      0      0      0      0      0      0      0
0      0.00000      .00000     00VAR1
   13    109    108    111      0      0      0      0      0      0      0
0    115.00000      .00000     00
  157    110     38      0      0      0      0      0      0      0      0
0    114.00000      .00000     00VAR1
   62    111    112    113      0      0      0      0      0      0      0
0      0.00000      .00000     00
  138    112      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     00+
  109    113    110      0      0      0      0      0      0      0      0
0      0.00000      .00000     00
```

```
  100   114     4     0     0     0     0     0     0     0     0
0      0.10000+001.00000    101
   52   115   116   118   105     0     0     0     0     0     0
0      0.00000     .00000    00
  166   116     0     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
  157   117    38     0     0     0     0     0     0     0     0
0    121.00000     .00000    00VAR1
   62   118   119   120     0     0     0     0     0     0     0
0      0.00000     .00000    00
  138   119     0     0     0     0     0     0     0     0     0
0      0.00000     .00000    00=
  109   120   117     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
  100   121     4     0     0     0     0     0     0     0     0
0      0.10000+002.00000    1010
  166   106     0     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
   88   105   106   107     0     0     0     0     0     0     0
0      0.00000     .00000    00
    8   101   102   104     0     0     0     0     0     0     0
0    122.00000     .00000    00
  100   103     4     0     0     0     0     0     0     0     0
0      0.30000+001.00000    103
  134   125   126     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
   97   126     0     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
   21    71    73    72     0     0     0     0     0     0     0
0    128.00000     .00000    00
    8   122   123   125     0     0     0     0     0     0     0
0      0.00000     .00000    00
   22   123   124     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
  102   124     0     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
  157   127    38     0     0     0     0     0     0     0     0
0      0.00000     .00000    00VAR1
   13   128   127   129     0     0     0     0     0     0     0
0    131.00000     .00000    00
  157   129    37     0     0     0     0     0     0     0     0
0      0.00000     .00000    00VAR2
  157   130    37     0     0     0     0     0     0     0     0
0      0.00000     .00000    00VAR2
   13   131   130   132     0     0     0     0     0     0     0
0    133.00000     .00000    00
  100   132     4     0     0     0     0     0     0     0     0
0      0.20000+002.00000    1020
   52   133   134   135    68     0     0     0     0     0     0
0      0.00000     .00000    00
  157   134    66     0     0     0     0     0     0     0     0
0      0.00000     .00000    00NAMED1
  166   135     0     0     0     0     0     0     0     0     0
0      0.00000     .00000    00
   87    64    65    67     0     0     0     0     0     0     0
0    137.00000     .00000    00
   86    65    67     0     0     0     0     0     0     0     0
0      0.00000     .00000    00LAB4
   86    66    68     0     0     0     0     0     0     0     0
```

```
0      0.00000      .00000     OONAMED1
    92     67     66     68      0      0      0      0      0      0      0
0      0.00000      .00000     OO
    88     68     69     70      0      0      0      0      0      0      0
0      0.00000      .00000     OO
   166     69      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO
   157    144     38      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOVAR1
   134    143    145      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO
    13    145    144    147      0      0      0      0      0      0      0
0    152.00000      .00000     OO
   157    146     38      0      0      0      0      0      0      0      0
0    150.00000      .00000     OOVAR1
    62    147    148    149      0      0      0      0      0      0      0
0      0.00000      .00000     OO
   138    148      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO+
   109    149    146      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO
   100    150      4      0      0      0      0      0      0      0      0
0      0.10000+001  .00000     101
   157    151     37      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOVAR2
    13    152    151    154      0      0      0      0      0      0      0
0    158.00000      .00000     OO
   157    153     37      0      0      0      0      0      0      0      0
0    157.00000      .00000     OOVAR2
    62    154    155    156      0      0      0      0      0      0      0
0      0.00000      .00000     OO
   138    155      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO+
   109    156    153      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO
   157    157     38      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOVAR1
    52    158    159    161    138      0      0      0      0      0      0
0      0.00000      .00000     OO
   157    159    136      0      0      0      0      0      0      0      0
0      0.00000      .00000     OONAMED2
   157    160     38      0      0      0      0      0      0      0      0
0    164.00000      .00000     OOVAR1
    62    161    162    163      0      0      0      0      0      0      0
0      0.00000      .00000     OO
   138    162      0      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO=
   109    163    160      0      0      0      0      0      0      0      0
0      0.00000      .00000     OO
   100    164      4      0      0      0      0      0      0      0      0
0      0.30000+001  .00000     103
    92    137    136    138      0      0      0      0      0      0      0
0    166.00000      .00000     OO
    88    138    139    143      0      0      0      0      0      0      0
0      0.00000      .00000     OO
    56    139    140    141      0      0      0      0      0      0      0
0      0.00000      .00000     OO
    83    140    141      0      0      0      0      0      0      0      0
0      0.00000      .00000     OOI
```

```
   33   141   142     0    26    26    26     0     0     0     0
0      0.00000      .00000     00
  157   142    25     0     0     0     0     0     0     0     0
0      0.00000      .00000     00COLOUR
   82   168   171     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
   70   169   170     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  161   170   172     0     0     0     0     0     0     0     0
0      0.00000      .00000     00VAR3
  134   174   176     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  157   175   170     0     0     0     0     0     0     0     0
0      0.00000      .00000     00VAR3
   13   176   175   177     0     0     0     0     0     0     0
0    178.00000      .00000     00
  100   177     3     0     0     0     0     0     0     0     0
0      0.33000+001  .00000     103.3
  134   185   187     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  157   186   170     0     0     0     0     0     0     0     0
0      0.00000      .00000     00VAR3
  100   188     3     0     0     0     0     0     0     0     0
0      0.66000+001  .00000     106.6
   13   187   186   188     0     0     0     0     0     0     0
0      0.00000      .00000     00
   71   178   179     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
   29   179   181   185     0     0     0     0     0     0     0
0      0.00000      .00000     00
  157   180    38     0     0     0     0     0     0     0     0
0    184.00000      .00000     00VAR1
   62   181   182   183     0     0     0     0     0     0     0
0      0.00000      .00000     00
  138   182     0     0     0     0     0     0     0     0     0
0      0.00000      .00000     00=
  109   183   180     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  157   184    37     0     0     0     0     0     0     0     0
0      0.00000      .00000     00VAR2
   92   166   165   167     0     0     0     0     0     0     0
0    189.00000      .00000     00
   19   167   168   174     0     0     0     0     0     0     0
0      0.00000      .00000     00
  160   171   169     0   172     0     0     0     0     0     0
0      0.00000      .00000     00
   33   172   173     0     3     3     0     0     0     0     0
0      0.00000      .00000     00
  157   173     3     0     0     0     0     0     0     0     0
0      0.00000      .00000     00FLOAT
  134   196   197     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  134   204   206     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  157   205    38     0     0     0     0     0     0     0     0
0      0.00000      .00000     00VAR1
   13   206   205   207     0     0     0     0     0     0     0
0      0.00000      .00000     00
  100   207     4     0     0     0     0     0     0     0     0
```

```
0      0.50000+001.00000     105
   29    198    200    204     0      0      0      0      0      0      0
0   208.00000      .00000     00
  134    214    216      0     0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    215     38      0     0      0      0      0      0      0      0
0      0.00000      .00000     00VAR1
   13    216    215    217     0      0      0      0      0      0      0
0      0.00000      .00000     00
  100    217      4      0     0      0      0      0      0      0      0
0      0.70000+001.00000     107
   29    208    210    214     0      0      0      0      0      0      0
0   218.00000      .00000     00
  134    224    226      0     0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    225     38      0     0      0      0      0      0      0      0
0      0.00000      .00000     00VAR1
   13    226    225    227     0      0      0      0      0      0      0
0      0.00000      .00000     00
  100    227      4      0     0      0      0      0      0      0      0
0      0.50000+001.00000     105
   29    218    220    224     0      0      0      0      0      0      0
0   228.00000      .00000     00
   29    228    229    230     0      0      0      0      0      0      0
0      0.00000      .00000     00
  157    231     38      0     0      0      0      0      0      0      0
0      0.00000      .00000     00VAR1
  134    230    232      0     0      0      0      0      0      0      0
0      0.00000      .00000     00
   13    232    231    233     0      0      0      0      0      0      0
0      0.00000      .00000     00
  100    233      4      0     0      0      0      0      0      0      0
0      0.30000+001.00000     103
   71    197    198      0     0      0      0      0      0      0      0
0   234.00000      .00000     00
  157    199     37      0     0      0      0      0      0      0      0
0   203.00000      .00000     00VAR2
   62    200    201    202     0      0      0      0      0      0      0
0      0.00000      .00000     00
  138    201      0      0     0      0      0      0      0      0      0
0      0.00000      .00000     00=
  109    202    199      0     0      0      0      0      0      0      0
0      0.00000      .00000     00
  100    203      4      0     0      0      0      0      0      0      0
0      0.10000+002.00000     1010
  157    209     37      0     0      0      0      0      0      0      0
0   213.00000      .00000     00VAR2
   62    210    211    212     0      0      0      0      0      0      0
0      0.00000      .00000     00
  138    211      0      0     0      0      0      0      0      0      0
0      0.00000      .00000     00=
  109    212    209      0     0      0      0      0      0      0      0
0      0.00000      .00000     00
  100    213      4      0     0      0      0      0      0      0      0
0      0.90000+001.00000     109
  157    219     37      0     0      0      0      0      0      0      0
0   223.00000      .00000     00VAR2
   62    220    221    222     0      0      0      0      0      0      0
0      0.00000      .00000     00
```

```
  138   221     0     0     0     0     0     0     0     0     0
0      0.00000      .00000     00=
  109   222   219     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  100   223     4     0     0     0     0     0     0     0     0
0      0.70000+001  .00000     107
  166   229     0     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
   52   234   235   236   189     0     0     0     0     0     0
0      0.00000      .00000     00
  166   235     0     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  166   236     0     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
   88   189   190   196     0     0     0     0     0     0     0
0    237.00000      .00000     00
  167   190   192     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  157   191    38     0     0     0     0     0     0     0     0
0    195.00000      .00000     00VAR1
   62   192   193   194     0     0     0     0     0     0     0
0      0.00000      .00000     00
  138   193     0     0     0     0     0     0     0     0     0
0      0.00000      .00000     00<
  109   194   191     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  100   195     4     0     0     0     0     0     0     0     0
0      0.10000+002  .00000     1010
  134   243   245     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  157   244    32     0     0     0     0     0     0     0     0
0      0.00000      .00000     00ROBOT
   13   245   244   246     0     0     0     0     0     0     0
0    247.00000      .00000     00
  157   246   239     0     0     0     0     0     0     0     0
0      0.00000      .00000     00J
  126   247   248     0     0     0     0     0     0     0     0
0    249.00000      .00000     00
  166   248     0     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
  126   249   250     0     0     0     0     0     0     0     0
0    251.00000      .00000     00
  157   250    38     0     0     0     0     0     0     0     0
0      0.00000      .00000     00VAR1
   97   251     0     0     0     0     0     0     0     0     0
0      0.00000      .00000     00
   88   237   238   243     0     0     0     0     0     0     0
0    253.00000      .00000     00
  127   238   239   240     0     0     0     0     0     0     0
0      0.00000      .00000     00
   83   239    26     0     0     0     0     0     0     0     0
0      0.00000      .00000     00J
  122   240   241   242    26     0     0     0     0     0     0
0      0.00000      .00000     00
  158   241    30    26     0     0     0     0     0     0     0
0      0.00000      .00000     10GREEN
  158   242    27    26     0     0     0     0     0     0     0
0      0.00000      .00000     10RED
  157   252    32     0     0     0     0     0     0     0     0
```

```
0       0.00000      .00000     00ROBOT
   86     165   167      0       0      0      0      0      0      0      0
0       0.00000      .00000     00NAMED3
   13     253   252    254       0      0      0      0      0      0      0
0       0.00000      .00000     00
   86     136   138      0       0      0      0      0      0      0      0
0       0.00000      .00000     00NAMED2
  158     254    30     26       0      0      0      0      0      0      0
0       0.00000      .00000     10GREEN
   19      16    17     42       0      0      0      0      0      0      0
0       0.00000      .00000     00
  160      39    36      0      40      0      0      0      0      0      0
0       0.00000      .00000     00
   33      40    41      0       4     20     21      0      0      0      0
0       0.00000      .00000     00
  157      41    19      0       0      0      0      0      0      0      0
0       0.00000      .00000     00INTEGER
   27      12     0      0      15      0      0      0      0      0      0
0       0.00000      .00000     00
```

13. REFERENCES.

[Bonet] R. Bonet et al. Ada Syntax Diagrams for Top-down Analysis, SIGPLAN Notices vol 16, no 9 September 1981.

[Evans 1982] A. Evans et al. Draft Revised DIANA Reference Manual, October 1982.

[Goodenough 1981] John B. Goodenough. The Ada Compiler Validation Capability, IEEE Computer June 1981.

[Goos 1981] G. Goos and Wm. A. Wulf. Diana Reference Manual, March 1981

[Harrison 1982] M. P. Harrison. Notes on changes in the Syntax of Ada, ADA UK News vol 3, no 2 October 1982.

[Military 1980] Military Standard Ada Programming Language, United States Department of Defence, Washington D.C. December 1980.

[Preliminary 1979] Preliminary Ada Reference Manual, SIGPLAN Notices vol 14, no 6 June 1979 part a.

[Reference 1982] Reference Manual for the ADA Programming Language, July 1982.

[Sperry Univac 1] Sperry Univac 1100, PLUS level 5R1, Programmer reference.
Sperry Univac 1100, PLUS level 4R1, Programmer reference.
Sperry Univac 1100, PLUS level 2R1, Programmer reference.

[Sperry Univac 2] Sperry Univac Series 1100, General Syntax Analyser GSA 1100 level 3R2, Programmer reference, 1982.

Sperry Univac Series 1100, General Syntax Analyser GSA 1100 level 3R1, Supplementary reference.
Sperry Univac Series 1100, General Syntax Analyser GSA 1100 level 2R1, Programmer reference, 1979.

[Steelman 1978] Steelman, Defence Advanced Research Projects Agency, Arlington, Virginia, 1979.

[Stenning 1981] Vic Stenning et al. The ADA environment: A Perspective, IEEE, Computer June 1981.

[Winkler 1981] J. F. H. Winkler. Differences between Preliminary and Final ADA, SIGPLAN Notices vol 16, no 11 November 1981.

[Wolfe 1981] Martin I. Wolfe. The ADA Language System, IEEE Computer June 1981.

14. BIBLIOGRAPHY.

ADA UK News.

J. G. P. BARNES. Programming in ADA, Addison-Wesley 1982.

Formal Definition of the ADA Programming Language, November 1980.

Ichbiah J. D. et al, Rationale for the design of the ADA programming Language, SIGPLAN Notices vol 14 no 6 June 1979 part b.

Proceedings of the ACM-SIGPLAN Symposium on the ADA Programming Language, Boston, Massachusetts, December 1980. SIGPLAN Notices vol 15 no 11 November 1980.

Proceedings of the Ada-TEC Conference on ADA. Arlington, Virginia, October 1982.

I. C. Pyle. The ADA Programming Language, Prentice Hall International 1981

P. Wegner. Programming with ADA: An introduction by means of graded examples, Prentice Hall International 1980.