A PARSER MODIFICATION OF THE EUCLID COMPILER: AUTOMATIC GENERATION OF SYNTAX ERROR RECOVERY

by

GRACE EVANS

B. M., University of Michigan, 1962

M. M., University of Michigan, 1963

A MASTER'S REPORT

submitted in partial fulfillment of the

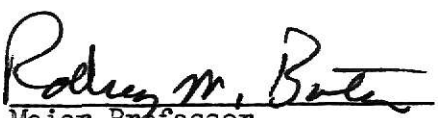
requirements for the degree

MASTER OF SCIENCE

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1982

Approved by:



Major Professor

ACKNOWLEDGEMENTS

With grateful thanks to Dr. Rodney Bates for his patience and help and to my husband, Bill, for his constant support and encouragement.

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

THIS IS AS RECEIVED FROM CUSTOMER.

TABLE OF CONTENTS

INTRODUCTION

Euclid is a relatively new programming language, designed to enable the construction of verifiable systems programs. [3] This project modified the Euclid compiler which was developed at the University of Toronto in 1977. [1] The original Toronto compiler provided minimal syntax error recovery in the parser, and this project was undertaken with three aims:

1) to provide more efficient syntax error recovery,
2) to provide a syntactically correct output token stream to the new pass, and
3) to demonstrate that the first two aims can be automatically generated.

Hartmann's error recovery scheme serves as the foundation for the generation of syntax error recovery. But in order to provide flexibility for potential future changes in the actual parsing program, the intent of this project is to provide the capability for automatic generation of the syntax error recovery, rather than being locked into any one particular language being parsed. The Euclid parser modules were modified with this end in mind.

Any parser written in S/SL consists of four parts: 1) an S/SL program, 2) an S/SL assember, 3) output from the S/SL assembler in table form, and 4) a table interpreter. The S/SL program contains the actual parsing procedures, and the other three parts implement its actions.

SYNTAX/SEMANTIC LANGUAGE

Syntax/Semantic Language (S/SL) is a modest programming language which was designed at the University of Toronto for use in the implementation of compilers. [4] It was developed from the use of syntax and semantic charts, which contain inputs, outputs, error messages, and semantic action calls all in the same diagram. [2] These charts could only be hand read and checked, and S/SL is a textual notation equivalent to them. Implemented by a translator/interpreter combination, as in the Euclid compiler, an S/SL program representing the syntax of a language can be computerized, which greatly simplifies compiler development.

LANGUAGE FEATURES

Much of the following background information is derived from a paper written by the authors of S/SL. [4]

S/SL is used for control only; the language contains no data or assignment statements. It consists only of sequences, repetitions (cycles), and choice of action statements which are combined to form rules, or procedures. A rule terminates with a semi-colon. Input, matching, or output of tokens and emission of error signals are the other features of S/SL. In some instances, an S/SL program may invoke semantic operations written in another language, such as Pascal, to manipulate data, but the parser pass of the Euclid compiler does not utilize a semantic mechanism, relying only on S/SL to perform the parsing and translation to a suitable output token stream for later passes of the Euclid compiler. Typical S/SL programs are, in fact, recursive descent

parsers written in S/SL.

The S/SL language dialect used by the Euclid parser has seven S/SL actions or statements:

1) the call action: "@" followed by a rule name indicates the calling of a rule, e.g., @TypeDeclaration,
2) the return action: "<" is found at the end of a rule and indicates that control should return to the point at which the rule was called,
3) the cycle (repeat) action: actions inside the braces are repeated until a cycle exit symbol (">") is encountered

```
{
  statements
}
```

4) the input (match) action: an input token name indicates the next input token read should match the token,
5) the input choice action: the brackets indicate a choice of actions, and the "=" indicates that the next input token should match one of the labels in this choice. The statements following the matched label and colon are then performed. An asterisk ("*") indicates an otherwise clause, which is taken if the current input token does not match a label

```
[=
    | label, label:
        actions
    | label:
        actions
      *:
    |     actions
]
```

6) the emit action: a dot (period) followed by an output token indicates that the token is to be emitted in the output stream (e.g.: .NotChecked), and

7) the exit action: ">" indicates an exit from the innermost cycle.

Input and output tokens which are used in the S/SL program are defined as sets by enumerating their names--for example:

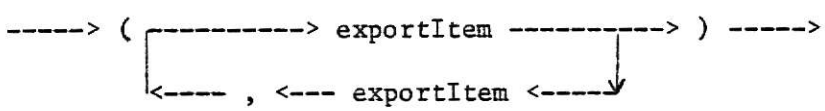
```
INPUTS:
    IDENT
    NUMBER
        .
        .
        .
    STRING;
```

The following example rule is from the S/SL program for the Euclid parser. ExportList parses a list of identifiers which are exported to

another module. In Euclid, in order for an identifier to be known outside of the scope of the module definition, the identifier must be explicitly exported. [3] This rule demonstrates the combination of actions and syntax in S/SL:

```
ExportList:
   "("@ExportItem
    {[=
         | ")":
              >
         | ",":
              @ExportItem
    ]}
```

The syntax diagram equivalent to the S/SL rule ExportList is as follows:



A Euclid export list consists of a list of one or more identifiers, separated by commas and enclosed in parentheses.

S/SL IMPLEMENTATION

The actual implementation of an S/SL program is accomplished in two steps. The first step converts, or assembles, the program into an intermediate language called S-code. In the Euclid parser, the conversion is accomplished by executing the S/SL assembler program with the parser S/SL program as its input.

A group of instructions, or operators, similar to a simple machine language is used for the translation of S/SL into the table of integers (S-code). Each S/SL statement is translated into S-code operators.

The S-code language contains the following nine operators and their respective operands:

OPERATOR		OPERAND
0	CALL	ADDRESS
1	RETURN	
2	REPEAT	ADDRESS
3	MERGE	ADDRESS
4	INPUT	INTOKEN
5	INPUTCHOICE	ADDRESS, NUMBER OF CHOICES INTOKEN, ADDRESS
6	EMIT	OUTTOKEN
7	ERROR	ERRORTOKEN
8	CHOICEEND	

The operand for the S-code operator CALL (0) is the location in the S-code for the rule being called. The operands for the operators REPEAT (2) and MERGE (3) indicate the location of the S-instruction to be executed next. The operand following the operator INPUT (4) is an input token matched to the input stream. EMIT (6) and ERROR (7) operands are token names emitted into the output and error streams, respectively. CHOICEEND (8) is emitted only for semantic choice failures; it is used in the parser to terminate a choice action but never emitted in the S-code.

The operator INPUTCHOICE (5) is followed by one operand, the address of the table. The table contains an odd number of values, depending upon the number of alternatives present in the choice action. The choice action is processed by emitting the operator INPUTCHOICE,

followed by the operand ADDRESS, which indicates the S-code location where the table of choice alternative labels is located. These labels are temporarily saved during assembly, while the actions for each choice alternative are emitted into the S-code, and then emitted as a table of choice labels. After all the choice alternatives have been processed and counted, the operand NUMBER OF CHOICES is emitted into the S-code followed by pairs of INTOKEN and ADDRESS equal to the NUMBER OF CHOICES. The INTOKEN is the previously-saved choice alternative label, and the ADDRESS is the location further back in the S-code where the actions for that alternative are located.

Using as an example the S/SL rule ExportList, the translation of the rule into S-code appears as follows:

TABLE LOCATION	S-CODE	ASSEMBLER CODE
376	4	INPUT
	29	LEFTPAREN
378	0	CALL
	400	EXPORTITEM
380	5	INPUTCHOICE
	390	TABLE
382	3	MERGE
	399	
384	3	MERGE
	397	
386	0	CALL
	400	EXPORTITEM
388	3	MERGE
	397	
390	2	TABLEBEGINS
391	30	RIGHTPAREN
	382	
393	34	COMMA
	386	
395	2	REPEAT
	382	
397	2	REPEAT
	380	
399	1	RETURN

The redundant S-instruction at location 384 results from a lack of any assembler optimization.

The second stage of implementing the S/SL program is called "table walking". The main part of the interpreter program consists of a large loop with a case statement, which contains a case label for each of the S-code operators. The interpreter "walks" through the S-code table by reading the operators, along with their respective operands, and interpreting their actions. In this manner, the statements of the original S/SL program are executed. In the case of the Euclid parser, the actions are executed upon the input stream from the previous pass, the lexical analyzer, and the token stream for the next pass is emitted.

HARTMANN ERROR RECOVERY

Prior to the modification detailed in this report, the syntax error recovery in the Euclid parser was minimal. In addition, the entire compilation process was halted in the event of even minor syntax errors. In order to improve upon these limitations and to maximize the detection of errors during a compilation attempt, the error recovery scheme of Hartmann was incorporated with the translation of the S/SL program into the table and its subsequent interpretation. However, it should be pointed out that the S/SL program itself was not changed.

The Hartmann error recovery scheme is derived directly from the syntax graphs of a particular language, just as the parsing procedures are. [3] In order to understand this recovery scheme, it is necessary to look first at the parser's input, which consists of tokens, possibly followed by semantic operands. The operands are ignored by the parser in the syntax analysis stage since they are concerned with semantics.

Hartmann's basic idea is to create key sets of tokens (deriving them from a syntax graph) which are used for recovery when a syntax error occurs.

The key sets represent tokens which may be encountered further downstream in the program from a given point in a syntax graph. Whenever a syntax error occurs, a key set is passed to a procedure which checks whether the current input token is contained in the set. If not, input tokens are skipped until one is found which is a member of the key set, and compilation may then continue. Parsing always resumes from the current point in the syntax graph. Thus, a valid path through the syntax graph is always traversed, regardless of input syntax errors, and

syntactically correct, though possibly meaningless, output is guaranteed.

Hartmann provides two basic rules for error recovery. The first requires that each set contain all tokens from which the compilation can resume, i.e., all the tokens which can be reached by taking any path from the current point in the syntax diagram. The second rule indicates that a check should be made at any branch (decision) in the syntax graph. A check procedure guarantees, by skipping input tokens if necessary, that the next input token is reachable from this decision point and, thus, insures that a decision is not based on an invalid input token. If the check procedure must skip tokens, it also emits an error message. The key set at that point should contain the next input token. This is the essence of the Hartmann scheme.

An implication of Hartmann's scheme is that a called procedure receives as input the key set which is reachable from its point of call. In Hartmann, a reachable set at any point is the union of tokens reachable within the current syntax diagram and those passed to the current procedure. The former can be computed at the time the parser is written or assembled; the latter is dynamic and known only at parser run time. When computing the reachable sets, a nonterminal in a syntax graph contributes only its handles (those tokens which may begin a string generated by a rule). As calls to procedures increase in depth, the size of the key sets increases, and conversely as procedures are completed, the key sets diminish in size. With this method of combining sets, the context in which a procedure is called has no direct influence on the procedure and the static part of its key sets.

According to Hartmann, "The process is so systematic that recursive

descent parsers with error recovery might be generated automatically from the language itself." [3] The key sets in Hartmann's Concurrent Pascal Compiler were hand generated, but it was his suggestion about automation which provided the inspiration for this project.

MODIFICATION

The modification for automatic generation of syntax error recovery required change in three parts of the implementation of S/SL: 1) the S-code language, 2) the assembler, and 3) the interpreter. Actual parser programs written in S/SL are not changed.

S-CODE

The first change involved three of the primitive operators in the S-code language. The addresses of the reachable key sets actually become a part of the S-code instructions as additional operands to the operators CALL, INPUT, and INPUTCHOICE. Each of these operators has a new operand which is the address of a reachable set for this particular point. Only the static part of the reachable set can appear in the S-code. INPUT and INPUTCHOICE operators also receive an S/SL program line number as an additional operand. This is an error message code to be used whenever an error message must be generated.

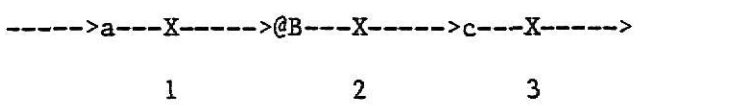
ASSEMBLER

The major changes were made to the S/SL assembler program. It was necessary to consider the key set generation from two perspectives. The first was concerned with generating sets of "reachable" tokens (symbols), i.e., those which can eventually be reached downstream from a given point in the syntax graph. The other type of set to be generated was the handle set (to be discussed later) of a rule, i.e., the tokens which may begin a string generated by the rule.

REACHABLE SETS. As observed by Hartmann, there are three constructs which are used in the writing of a program: the sequence, the cycle (loop), and the choice (branch). These are exactly the three constructs also available in S/SL. These constructs are used separately or in combination, and it is the combination which complicates the syntax analysis of a program. Central to the issue of adding tokens to the reachable sets is the assembler's ability to determine the current nesting level, and this is accomplished through stack action. The extant assembler program contained choice, exit, merge, and cycle stacks for semantic control, and these were augmented by the addition of a sequence stack.

As the processing of each procedure begins, the sequence stack is empty, and a new sequence is pushed. Each time one of the three modified operators is generated in the S/SL assembler program, a new empty set is obtained from the newly-created data structure, SETTABLE, and its set label is added as an operand to the primitive operator and attached to the list of set labels stored on the sequence stack. With every new encounter, the top sequence stack entry receives another set label, and each of the earlier set labels also has the input token (or, in the case of a CALL, a set label) added to its set. In this manner an input token or procedure call set label is added to all those reachable key sets which are behind it in the list on the sequence stack.

Consider the following sequence:



where the small letters (a,c) represent input actions, the capital letter (B) preceded by the "@" represents a procedure call, and the X's each represent a point where a reachable set must be computed, i.e., following an INPUT or CALL operator. The numbers are set labels which correspond to the points along the sequence. The operators emitted by the assembler with their operands are:

INPUT, a, (address), 1

CALL, B, 2

INPUT, c, (address), 3

If "a" is the first sequence item processed, its reachable set is empty initially. After "@B" is processed, the set (labelled 1) for "a" contains the handle set of "B". After "c" is processed, the sets appear as follows:

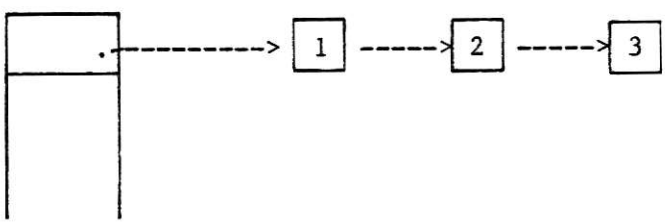
SET 1 (B-handles, c)

SET 2 (c)

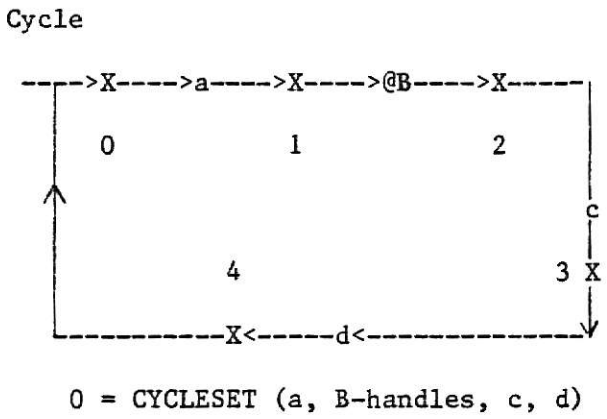
SET 3 ( )

and the sequence stack has one entry with a list of three labels:

Sequence Stack



A cycle (loop) is processed in similar fashion, except that an additional set (CYCLESET) is needed at the cycle's beginning. This cycle set (whose label is stored on the cycle stack) receives all the reachable tokens in a cycle, as in the following example:



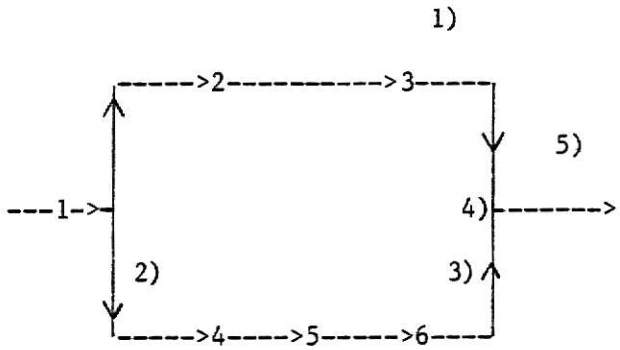
0 = CYCLESET (a, B-handles, c, d)

The cycleset is never put directly into the S-code; but at the end of a cycle, it is added to any sets which are in the list at the top level of the sequence stack.

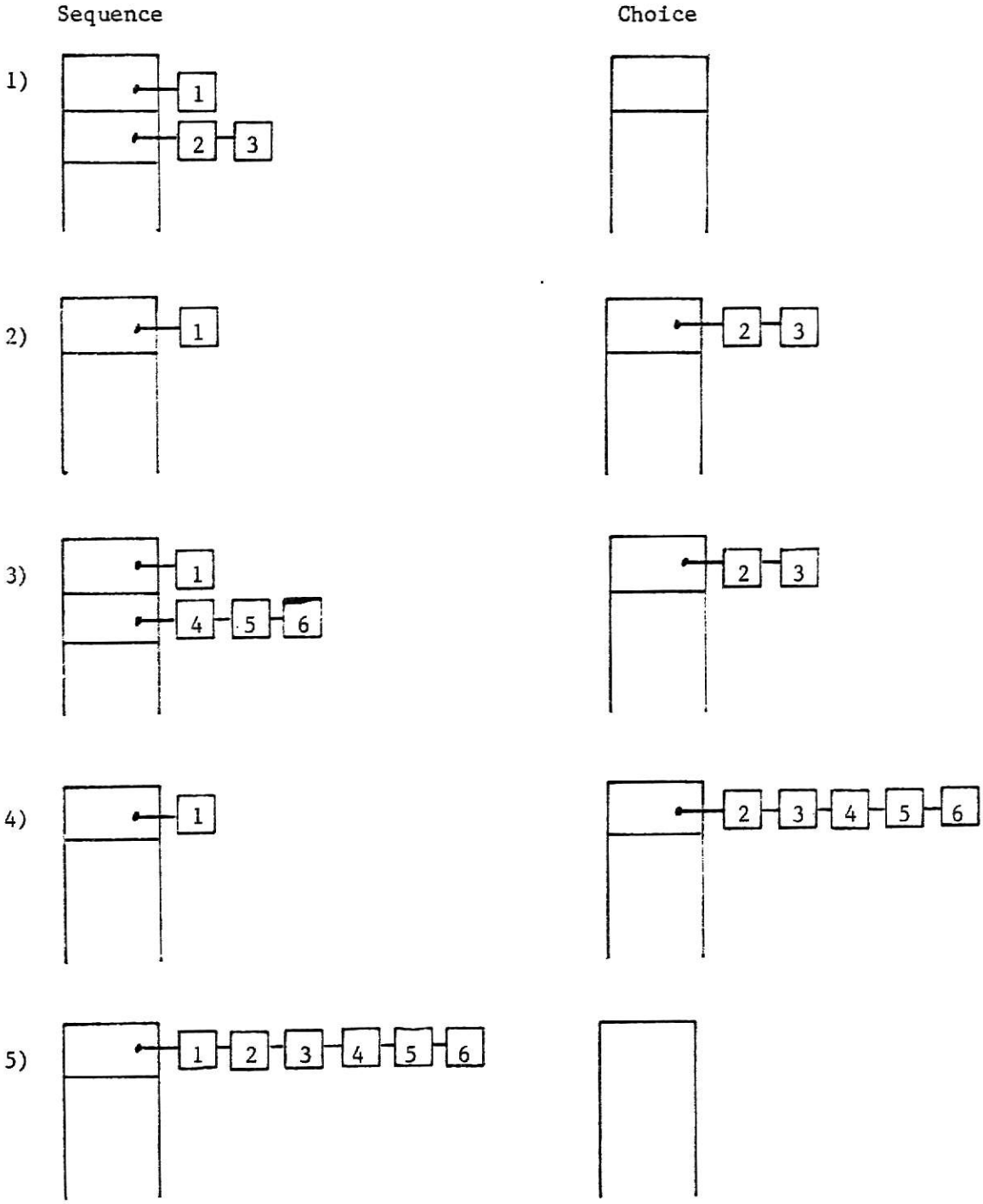
A choice (branch) is processed by treating each alternative path as a separate sequence, pushing and popping each one onto the sequence stack. At the end of processing a single choice alternative, the list of set labels for it is moved temporarily to the choice stack entry in order to process the next alternative. After all alternatives have been processed (allocating and adding tokens to the sets), the entire list (consisting of list segments of set labels from each alternative) is moved back to the sequence stack entry for the sequence containing the choice.
The following series of stack snapshots shows the process. In the Choice Path diagram, numbers in the diagram paths are set labels.

Numbers followed by ")" are snapshot numbers (see Stack Action diagram which follows the Choice Path diagram below).

Choice Paths



Stack Action

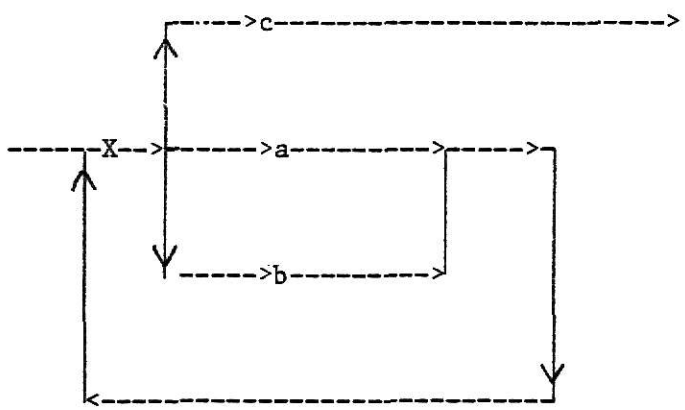


When a choice action is combined with a cycle, the set generation becomes more complicated. If choice alternatives are contained inside a cycle, the only path(s) out of the cycle are those which terminate with an exit (">") symbol. Although these paths are textually inside the cycle, they are outside the cycle in the corresponding syntax graph. The sets along these paths do not have the CYCLESET added to them, as the following example demonstrates.

S/SL Rule Segment

```
{[=
     | c:---   > ;
     | a:--------;
     | b:--------;
 ]}
```

Cycle With Choice Inside



In this example, "a" and "b" are not reachable from "c"; so, the sets along the "c" path do not include the CYCLESET. However, if the cycle itself is nested inside another cycle, all inner paths (a,b,c) may be

reachable from any other.

HANDLE SETS. In addition to the reachable sets, it is necessary to generate procedure handle sets, which are added to other reachable sets (instead of adding tokens) each time a procedure call is assembled. The handle set consists of those tokens which may begin a string generated by the procedure. When a procedure is initially referenced, whether at a CALL statement or at the actual procedure declaration, a set label is allocated for its handle set.

Procedure handles are generated by initially turning on a Boolean flag in the sequence stack entry to indicate that the procedure is exposed (i.e., the current point in its syntax graph can derive empty). If the procedure begins with a simple sequence, the flag is turned off when a token or another non-empty handle set is added to the procedure's handle set.

It is a bit more complex if a cycle or choice construct is involved. For either, it is necessary to set initial and final exposed flags. If any path contained in a cycle or a choice is exposed at the end of its processing, the final flag for either is true, and the process of adding to handle sets continues until the initial sequence's exposed flag can be turned off.

SET IMPLEMENTATION. The Euclid subset which was used to write the parser does not support a conventional set implementation. However, a Powerset of fifteen bits is available, and any number of these sets can be put together in an array to supply the required number of set members. The value of each bit in the Powerset goes up by a power of two

(i.e., 1, 2, 4, 8, 16, etc.), providing each Powerset with a unique integer value--depending on the bits turned on. Thus, it is possible to examine individual bits or to compare entire Powersets for equality.

It might also be pointed out that the generation of error recovery sets in the assembler produces two different types of members in each set. The first type is actual tokens, or bits, in a set, but the second is only a list of set labels for other sets in the set table. Before the assembler can emit the S-code table, it is necessary to combine the list of set labels into a pure set using a recursive transitive closure algorithm.

CLOSURE. The recursive closure algorithm closes each set table entry. An entry consists of a pure set of bits (8 Powersets) and, possibly, a list of set labels, which results from adding handle sets and cycle sets (in set label form) to reachable sets.

In order to understand the algorithm, each set table entry should be thought of as the parent node, or root, of a graph and its list of set labels as its children. As each parent node is processed, its set visiting flag is turned on. A loop is executed as long as there are children (i.e., set labels) available whose own set visiting flags are not turned on. Each available child, in turn, is deleted from the parent's list of set labels. A child whose visiting flag is turned on is bypassed.

At this point, the closure algorithm calls itself recursively, passing the current child as a parameter which then becomes a parent (root) node, and the process begins again. At the return of each

recursive call, the union of the child's set with the parent's set is created, and the child's list of set labels is added to the parent's list.

The effect of this algorithm on each set table entry is to close completely the parent set, and, as a side effect, to possibly close some of the children's sets.

DUPLICATE SETS AND SET LABEL TRANSLATION. After the S/SL identifier definitions and procedures have been processed and set closure obtained, it is necessary to check for duplicate set generation, eliminate the duplicates using a hash table, and allocate space at the end of the S-code table for all the sets. Each set table entry is now a pure set of bits, consisting of multiple Powersets. The integer values of each Powerset must be totalled and the sum hashed into a table in order to identify duplicate sets.

If a set is not a duplicate, space at the end of the S-code table is allocated for it, and the S-code address is saved in the set table entry. In the case of a duplicate set, the S-code address for the previously-allocated set is located and stored with the set table entry. When the actual S-code table is emitted, each set label operand is then translated into a real S-code address.

SUMMARY. The assembler, then, accomplishes the following:

1) generation of the S-code and static sets (reachable and handle),
2) closure of the sets,
3) elimination of duplicate sets,
4) translation of set labels to actual S-code addresses,
5) resolution of rule addresses, and
6) emission of set values into the S-code.

INTERPRETER

Minor changes were made to the S-code interpreter program to complete the error recovery modification.

The integer table of S-code is declared as a constant in the interpreter, which then alternately reads the input token stream and "walks" the table. It was necessary to change the interpreter program to include the additional operands for CALL, INPUT, and INPUTCHOICE.

The call stack, which implements the S/SL procedure call and return, was given an extra field to hold sets, and it became the final link in the automatic set generation. An empty set is initially placed on the bottom of the stack. As each CALL operator is processed, its reachable set operand is converted to an actual set. The union of this set and the set on top of the call stack is created, and the new set is pushed on the stack. Each procedure return pops the call stack a level, exposing the previous set.

When an incoming token is not matched to an expected one, the syntax error procedure has been changed to receive the set operand following the

INPUT operator. This set address is converted to a real set, and a union is created between the operator's reachable set and the set on the top entry of the call stack. An error message is emitted, incoming tokens are skipped until one is found which is contained in the union of the two sets, and normal parsing continues. In the case of an INPUTCHOICE operator, a check is performed on the union of the two sets to determine whether the incoming token is found in the set. If not, the error message and skipping process is again performed.

FUTURE WORK

Further work on the assembler is needed in order to correctly compute the "derived-empty" property of a rule. In the present modification, a rule is presumed to derive empty unless its processing proves otherwise. In other words, if a call is made on an unprocessed rule, the exposed flag for the calling rule is not turned off even though the handle set is added to the set generated by the call statement. To compute the property accurately would require another pass, in addition to the present two, and another complex algorithm similar to that which computes and closes the sets.

Another area for future work is automatic generation of error message text. At present, error messages consist of the S/SL program's line number, the broad category of error type (e.g., syntax, premature end-of-file, and call stack overflow), and the input program's line number. The generation of specific syntax error messages from the S/SL program would be a significant improvement over the existent handling of error messages.

CONCLUSION

With the Hartmann error recovery scheme as a framework, it has been possible to automatically generate syntax error recovery in a systematic way. The Euclid compiler can now continue the parsing process, despite syntactical errors which would have previously caused it to abort compilation entirely. In addition, the token output to the next pass is syntactically correct, allowing the next pass to begin its analysis.

More important, however, is the fact that this automatic syntax error recovery works for any S/SL program, without additional modification to the translator/interpreter implementation and without awareness of error recovery on the part of the S/SL program writer.

REFERENCES CITED

[1] April Euclid Translator. Toronto Euclid Compiler (second phase), Computer Systems Research Group, University of Toronto, and Special Systems Division, I. P. Sharp Associates Limited, June 1979.

[2] Cordy, James R., Richard C. Holt, and David B. Wortman. Semantic Charts: A Diagrammatic Approach to Semantic Processing. SIGPLAN Symposium on Compiler Construction (August 1979), 39-49.

[3] Hartmann, Alfred C. A Concurrent Pascal Compiler for Minicomputers. Ph.D. dissertation, California Institute of Technology (1976), 27-40.

[4] Holt, R. C., J. R. Cordy, and D. B. Wortman. Introduction to S/SL: Syntax/Semantic Language. Technical Report of Computer Systems Research Group, University of Toronto, March 1980.

[5] Lampson, B. W. et al. Report on the Programming Language Euclid. SIGPLAN Notices 12, 2 (February 1977), 1-79.

APPENDICES

```
1
2
3   %       Project Euclid
4   %
5   %       Euclid Parser Tables V1.24
6   %       Author: J.R. Cordy
7   %       Reader: D.R. Crowe
8   %       Date:   27 Feb 1979
9
10
11
12  Inputs :
13
14      tIdent      'Ident'
15      tNumber     'Number'
16      tOctalNumber 'OctalNumber'
17      tHexNumber  'HexNumber'
18      tString     'String'
19      tExString   'ExString'
20      tMDString   'MDString'
21      tChar       'Char'
22      tExChar     'ExChar'
23      tMDChar     'MDChar'
24      tComment
25      tCommentStart
26      tCommentMiddle
27      tCommentEnd
28      tCodeStart  'CodeStart'
29      tCodeMiddle 'CodeMiddle'
30      tCodeEnd    'CodeEnd'
31      tIllegal
32      tNewLine    '<NL>'
33      tEndOfFile
34      tPlus       '+'
35      tMinus      '-'
36      tAsterisk   '*'
37      tEqual      '='
38      tLessThan   '<'
39      tGreaterThan '>'
40      tLessEqual  '<='
41      tGreaterEqual '>='
42      tImplies    '->'
43      tLeftParen  '('
44      tRightParen ')'
45      tPeriod     '.'
46      tDoublePeriod '..'
47      tAssign     ':='
48      tComma      ','
49      tSemicolon  ';'
50      tColon      ':'
51      tUpArrow    '^'
52      tLabel      '=>'
```

```
53        tAbstraction 'abstraction'
54        tAligned     'aligned'
55        tAll         'all'
56        tAnd         'and'
57        tAny         'any'
58        tArray       'array'
59        tAssert      'assert'
60        tAt          'at'
61        tBegin       'begin'
62        tBind        'bind'
63        tBits        'bits'
64        tBound       'bound'
65        tCase        'case'
66        tCheckable   'checkable'
67        tChecked     'checked'
68        tCode        'code'
69        tCollection  'collection'
70        tConst       'const'
71        tConverter   'converter'
72        tCounted     'counted'
73        tDecreasing  'decreasing'
74        tDefault     'default'
75        tDependent   'dependent'
76        tDiv         'div'
77        tElse        'else'
78        tElseif      'elseif'
79        tEnd         'end'
80        tExit        'exit'
81        tExports     'exports'
82        tExternal    'external'
83        tFinally     'finally'
84        tFor         'for'
85        tForward     'forward'
86        tFrom        'from'
87        tFunction    'function'
88        tIf          'if'
89        tImports     'imports'
90        tIn          'in'
91        tInclude     'include'
92        tInitially   'initially'
93        tInline      'inline'
94        tInvariant   'invariant'
95        tLoop        'loop'
96        tMachine     'machine'
97        tMod         'mod'
98        tModule      'module'
99        tNot         'not'
100       tOf          'of'
101       tOr          'or'
102       tOtherwise   'otherwise'
103       tPacked      'packed'
104       tParameter   'parameter'
```

```
105         tPervasive  'pervasive'
106         tPost       'post'
107         tPre        'pre'
108         tProcedure  'procedure'
109         tReadonly   'readonly'
110         tRecord     'record'
111         tReturn     'return'
112         tReturns    'returns'
113         tSet        'set'
114         tThen       'then'
115         tThus       'thus'
116         tTo         'to'
117         tType       'type'
118         tUnknown    'unknown'
119         tVar        'var'
120         tWhen       'when'
121         tWith       'with'
122         tXor        'xor';
123
124
125     Outputs :
126
127         aIdent
128         aNumber
129         aString
130         aMDString
131         aChar
132         aMDChar
133         aCodeUnit
134         aStandardComponent
135         aLegalitySpecifier
136         aNewLine
137         aEndOfFile
138         aAbstraction
139         aAdd
140         aAligned
141         aAll
142         aAnd
143         aAny
144         aArray
145         aAssert
146         aAssign
147         aAt
148         aBegin
149         aBind
150         aBits
151         aBound
152         aCase
153         aCheckable
154         aChecked
155         aCode
156         aCollection
```

```
157     aConst
158     aConstType
159     aConverter
160     aCountMax
161     aCounted
162     aDecreasing
163     aDefault
164     aDiv
165     aElse
166     aEndBegin
167     aEndBind
168     aEndCase
169     aEndCode
170     aEndEnum
171     aEndExports
172     aEndExpression
173     aEndIdent
174     aEndIf
175     aEndImports
176     aEndLabel
177     aEndLabels
178     aEndLoop
179     aEndModule
180     aEndParms
181     aEndRecord
182     aEndSubs
183     aEndValues
184     aEndWith
185     aEnum
186     aEqual
187     aExit
188     aExports
189     aExternal
190     aField
191     aFinally
192     aFor
193     aForward
194     aFunction
195     aGreater
196     aGreaterEqual
197     aIf
198     aImply
199     aImports
200     aIn
201     aInfixAnd
202     aInfixCompare
203     aInfixImply
204     aInfixOr
205     aInitial
206     aInitially
207     aInline
208     aInvariant
```

```
209         aLabels
210         aLess
211         aLessEqual
212         aLoop
213         aMDModule
214         aMDRecord
215         aMinus
216         aMod
217         aModule
218         aModuleIdent
219         aMultiply
220         aNot
221         aNotChecked
222         aNotEqual
223         aNotIn
224         aOr
225         aOtherwise
226         aPacked
227         aParameter
228         aParens
229         aParmType
230         aParms
231         aPervasive
232         aPointer
233         aPost
234         aPre
235         aProcedure
236         aReadonly
237         aRecord
238         aReturn
239         aReturnValue
240         aReturns
241         aSet
242         aSubs
243         aSubtract
244         aTo
245         aType
246         aTypeName
247         aUnknown
248         aValues
249         aVar
250         aVarType
251         aWhen
252         aWith
253         aXor;
254
255
256     EmitOps :
257
258         oEmitIdent
259         oEmitChar
260         oEmitNumber
```

```
261         oEmitString
262         oEmitCode
263         oEmitLine;
264     _|_
265
266
267     %    Programs
268
269     Program :
270
271         oEmitLine @Pervasive 'type' @TypeDeclaration ';' .aEndOfFile;
272
273
274
275     %    Modules
276
277     ModuleBody :
278
279         [=
280             | 'Ident':
281                 .aModuleIdent .aIdent oEmitIdent
282             | *:
283         ]
284         '<NL>' oEmitLine
285         @ImportClauses
286         @ExportClause
287         [=
288             | 'not':
289                 'checked' .aNotChecked ';'
290                 oEmitLine
291             | 'checked':
292                 .aChecked ';'
293                 oEmitLine
294             | *:
295         ]
296         @Declarations
297         [=
298             | 'initially':
299                 .aInitially @RoutineDefinition ';'
300                 oEmitLine
301             | *:
302         ]
303         [=
304             | 'abstraction':
305                 .aAbstraction 'function' @FunctionDeclaration ';'
306                 'invariant' oEmitLine .aInvariant '(' @Expression ')' ';'
307                 oEmitLine
308             | 'invariant':
309                 .aInvariant '(' @Expression ')' ';'
310                 oEmitLine
311             | *:
312         ]
```

```
313         [=
314             | 'finally':
315                 .aFinally @RoutineDefinition ';'
316                 oEmitLine
317             | *:
318         ];
319
320
321
322     ImportClauses :
323
324         {[=
325             | 'imports':
326                 .aImports '(' @ImportItem
327                 {[=
328                     | ')':
329                         >
330                     | ',':
331                         @ImportItem
332                 ]}
333                 .aEndImports
334                 [=
335                     | 'thus':
336                         '(' @ThusItem
337                         {[=
338                             | ')':
339                                 >
340                             | ',':
341                                 @ThusItem
342                         ]}
343                     | *:
344                 ]
345                 ';' oEmitLine
346             | *:
347                 >
348         ]};
349
350
351
352     ImportItem :
353
354         @Pervasive @BindingCondition 'Ident' .aIdent oEmitIdent;
355
356
357
358     Pervasive :
359
360         [=
361             | 'pervasive':
362                 .aPervasive
363             | *:
364         ];
```

```
365
366
367
368     BindingCondition :
369
370         [=
371             | 'var':
372                 .aVar
373             | 'readonly':
374                 .aReadonly
375             | 'const':
376                 .aConst
377             | *:
378         ];
379
380
381
382     ThusItem :
383
384         [=
385             | 'pervasive':
386             | *:
387         ]
388         [=
389             | 'var':
390             | 'readonly':
391             | 'const':
392             | *:
393         ]
394         'Ident';
395
396
397
398     ExportClause :
399
400         [=
401             | 'exports':
402                 .aExports @ExportList .aEndExports ';'
403                 oEmitLine
404             | *:
405         ];
406
407
408
409     ExportList :
410
411         '(' @ExportItem
412         {[=
413             | ')':
414                 >
415             | ',':
416                 @ExportItem
```

```
417         ]};
418
419
420
421     ExportItem :
422
423         [=
424             | ':=':
425                 .aAssign
426             | '=':
427                 .aEqual
428             | '^':
429                 .aPointer
430             | 'Ident':
431                 .aIdent oEmitIdent
432                 [=
433                     | '..':
434                         .aTo 'Ident' .aIdent oEmitIdent
435                     | *:
436                         @WithClause
437                 ]
438             | *:
439                 @BindingCondition
440                 'Ident' .aIdent oEmitIdent @WithClause
441         ];
442
443
444
445     WithClause :
446
447         [=
448             | 'with':
449                 .aWith @ExportList .aEndWith
450             | *:
451         ];
452
453
454     %    Declarations
455
456     Declarations :
457
458         {
459             [=
460                 | 'var':
461                     @VariableDeclaration
462                 | 'bind':
463                     @VariableBinding
464                 | 'assert':
465                     @AssertStatement
466                 | 'pervasive':
467                     .aPervasive
468                     [=
```

```
469                     | 'const':
470                         @ConstantDeclaration
471                     | 'type':
472                         @TypeDeclaration
473                     | 'converter':
474                         @ConverterDeclaration
475                     | 'inline':
476                         @InlineProcedureDeclaration
477                     | 'procedure':
478                         @ProcedureDeclaration
479                     | 'function':
480                         @FunctionDeclaration
481                 ]
482             | 'const':
483                 @ConstantDeclaration
484             | 'type':
485                 @TypeDeclaration
486             | 'converter':
487                 @ConverterDeclaration
488             | 'inline':
489                 @InlineProcedureDeclaration
490             | 'procedure':
491                 @ProcedureDeclaration
492             | 'function':
493                 @FunctionDeclaration
494             | ';':
495             | *:
496                 ';' >
497         ]
498         ';' oEmitLine
499     };
500
501
502
503 VariableDeclaration :
504
505     .aVar 'Ident' .aIdent oEmitIdent
506     [=
507         | '(':
508             'at' .aAt @Expression ')'
509         | *:
510             {[=
511                 | ',':
512                     'Ident' .aIdent oEmitIdent
513                 | *:
514                     >
515             ]}
516     ]
517     ':' oEmitLine .aVarType @TypeDefinition
518     [=
519         | ':=':
520             .aInitial @Expression
```

```
521             | *:
522         ];
523
524
525
526     VariableBinding :
527
528         .aBind
529         [=
530             | '(':
531                 @RenameVariable
532                 {[=
533                     | ')':
534                         >
535                     | ',':
536                         @RenameVariable
537                 ]}
538             | *:
539                 @RenameVariable
540         ]
541         .aEndBind;
542
543
544
545     RenameVariable :
546
547         @VarBindingCondition 'Ident' .aIdent oEmitIdent 'to' .aTo 'Ident'
548             @Variable;
549
550
551
552     VarBindingCondition :
553
554         [=
555             | 'var':
556                 .aVar
557             | 'readonly':
558                 .aReadonly
559             | *:
560         ];
561
562
563
564     ConstantDeclaration :
565
566         .aConst 'Ident' .aIdent oEmitIdent
567         {[=
568             | ',':
569                 'Ident' .aIdent oEmitIdent
570             | *:
571                 >
572         ]}
```

```
573         [=
574             | ':=':
575                 .aInitial @Expression
576             | ':':
577                 oEmitLine .aConstType @TypeDefinition
578                 ':='
579                 [=
580                     | '(':
581                         @StructuredConstant ')'
582                     | *:
583                         .aInitial @Expression
584                 ]
585         ];
586
587
588
589     StructuredConstant :
590
591         .aValues
592         {
593             [=
594                 | '(':
595                     @StructuredConstant ')'
596                 | *:
597                     @Expression
598             ]
599             [=
600                 | ',':
601                 | *:
602                     >
603             ]
604         }
605         .aEndValues;
606
607
608
609     ConverterDeclaration :
610
611         .aConverter 'Ident' .aIdent oEmitIdent '('
612         [=
613             | 'Ident':
614                 .aIdent oEmitIdent
615             | 'procedure':
616                 .aProcedure
617             | 'function':
618                 .aFunction
619         ]
620         ')' 'returns' .aReturns 'Ident' .aIdent oEmitIdent;
621
622
623
624     TypeDeclaration :
```

```
625
626         .aType 'Ident' .aIdent oEmitIdent
627         [=
628             | '(':
629                 .aParms @TypeFormal
630                 {[=
631                     | ')':
632                         >
633                     | ',':
634                         @TypeFormal
635                 ]}
636                 .aEndParms
637             | *:
638         ]
639         '=' '<NL>'
640         [=
641             | 'pre':
642                 [=
643                     | '(':
644                         oEmitLine .aPre @Expression ')'
645                     | *:
646                 ]
647                 ';'
648             | *:
649         ]
650         oEmitLine
651         [=
652             | 'forward':
653                 .aForward
654             | *:
655                 @TypeDefinition
656         ];
657
658
659
660     TypeFormal :
661
662         @Pervasive
663         [=
664             | 'const':
665             | *:
666         ]
667         'Ident' .aIdent oEmitIdent
668         {[=
669             | ',':
670                 'Ident' .aIdent oEmitIdent
671             | *:
672                 >
673         ]}
674         ':' .aParmType @IndexType;
675
676
```

```
677
678   InlineProcedureDeclaration :
679
680       .aInline
681       [=
682           | 'procedure':
683               @ProcedureDeclaration
684           | 'function':
685               @FunctionDeclaration
686       ];
687
688
689   %    Types
690
691   TypeDefinition :
692
693       [=
694           | '(':
695               @EnumeratedTypeDefinition
696           | 'array':
697               .aArray @IndexType 'of' @TypeDefinition
698           | 'record':
699               .aRecord @RecordBody @RecordTrailer
700           | 'module':
701               .aModule @ModuleBody @ModuleTrailer
702           | 'set':
703               .aSet 'of' @IndexType
704           | 'machine':
705               'dependent'
706               [=
707                   | 'record':
708                       .aMDRecord
709                       [=
710                           | 'aligned':
711                               'mod' .aAligned @Expression
712                           | *:
713                       ]
714                       @MDRecordBody @RecordTrailer
715                   | 'module':
716                       .aMDModule @ModuleBody @ModuleTrailer
717               ]
718           | 'collection':
719               .aCollection @CollectionTypeDefinition
720           | 'counted':
721               .aCollection .aCounted
722               [=
723                   | 'collection':
724                   | *:
725                       .aCountMax @Expression 'collection'
726               ]
727               @CollectionTypeDefinition
728           | 'checkable':
```

```
729             'collection' .aCollection .aCheckable
730             @CollectionTypeDefinition
731         | '^':
732             .aPointer 'Ident' @Variable
733         | 'packed':
734             .aPacked
735             [=
736                 | 'array':
737                     .aArray @IndexType 'of' @TypeDefinition
738                 | 'record':
739                     .aRecord @RecordBody @RecordTrailer
740                 | 'module':
741                     .aModule @ModuleBody @ModuleTrailer
742                 | 'set':
743                     .aSet 'of' @IndexType
744                 | 'machine':
745                     'dependent' 'module' .aMDModule @ModuleBody
746                         @ModuleTrailer
747             ]
748         | *:
749             @NamedOrRangeTypeDefinition
750     ];
751
752
753
754 IndexType :
755
756     [=
757         |'(':
758             @EnumeratedTypeDefinition
759         | *:
760             @NamedOrRangeTypeDefinition
761     ];
762
763
764
765 EnumeratedTypeDefinition :
766
767     .aEnum 'Ident' .aIdent oEmitIdent
768     {[=
769         | ')':
770             >
771         | ',':
772             'Ident' .aIdent oEmitIdent
773     ]}
774     .aEndEnum;
775
776
777
778 NamedOrRangeTypeDefinition :
779
780     [=
```

```
781             | 'Ident':
782                 @Variable
783                 [=
784                     | '..':
785                         .aEndExpression
786                     | *:
787                         [=
788                             | '+':
789                                 @Term .aAdd @PartialSum
790                             | '-':
791                                 @Term .aSubtract @PartialSum
792                             | 'xor':
793                                 @Term .aXor @PartialSum
794                             | '*':
795                                 @Factor .aMultiply @PartialSum
796                             | 'div':
797                                 @Factor .aDiv @PartialSum
798                             | 'mod':
799                                 @Factor .aMod @PartialSum
800                             | *:
801                                 .aTypeName >>
802                         ]
803                         '..'
804                 ]
805                 .aTo @Sum
806             | *:
807                 @Sum '..' .aTo @Sum
808         ];
809
810
811
812     RecordBody :
813
814         @DeclarationsInRecord
815         [=
816             | 'case':
817                 .aCase 'Ident' .aIdent oEmitIdent
818                 [=
819                     | 'default':
820                         .aDefault @Expression
821                     | *:
822                 ]
823                 'of' '<NL>'
824                 oEmitLine
825                 {[=
826                     | 'end':
827                         'case' >
828                     | 'otherwise':
829                         '=>' '<NL>'
830                         .aOtherwise @DeclarationsInRecord
831                         'end' 'case' >
832                     | *:
```

```
833                     .aLabels @CaseLabel
834                     {[=
835                         | ',':
836                             @CaseLabel
837                         | *:
838                             >
839                     ]}
840                     .aEndLabels
841                     '=>' '<NL>'
842                     @DeclarationsInRecord
843                     'end' .aEndLabel @CaseLabel ';'
844                     oEmitLine
845                 ]}
846                 .aEndCase ';'
847                 oEmitLine
848             | *:
849         ];
850
851
852
853     DeclarationsInRecord :
854
855         {
856             oEmitLine
857             [=
858                 | 'var':
859                     @VariableDeclaration
860                 | 'const':
861                     @ConstantDeclaration
862                 | 'pervasive':
863                     .aPervasive 'const' @ConstantDeclaration
864                 | 'assert':
865                     @AssertStatement
866                 | ';':
867                 | *:
868                     ';' >
869             ]
870             ';'
871         };
872
873
874
875     RecordTrailer :
876
877         'end' .aEndRecord
878         [=
879             | 'record':
880             | 'Ident':
881                 .aEndIdent .aIdent oEmitIdent
882         ];
883
884
```

```
885
886     ModuleTrailer :
887
888         'end' .aEndModule
889         [=
890             | 'module':
891             | 'Ident':
892                 .aEndIdent .aIdent oEmitIdent
893         ];
894
895
896
897     MDRecordBody :
898
899         {
900             oEmitLine
901             [=
902                 | 'var':
903                     .aVar 'Ident' .aIdent oEmitIdent
904                     '(' 'at' .aAt @Expression
905                     [=
906                         | 'bits':
907                             .aBits @Sum '..' .aTo @Sum
908                         | *:
909                     ]
910                     ')' ':' .aVarType @TypeDefinition
911                     [=
912                         | ':=':
913                             .aInitial @Expression
914                         | *:
915                     ]
916                 | 'const':
917                     @ConstantDeclaration
918                 | 'pervasive':
919                     .aPervasive 'const' @ConstantDeclaration
920                 | 'assert':
921                     @AssertStatement
922                 | ';':
923                 | *:
924                     ';' >
925             ]
926             ';'
927         };
928
929
930
931     CollectionTypeDefinition :
932
933         'of' .aVarType @TypeDefinition
934         [=
935             | 'in':
936                 .aIn 'Ident' @Variable
```

```
937             | *:
938         ];
939
940
941     %    Routines
942
943     ProcedureDeclaration :
944
945         .aProcedure 'Ident' .aIdent oEmitIdent @FormalParameters '='
946             '<NL>'
947         @RoutineDefinition;
948
949
950
951     FunctionDeclaration :
952
953         .aFunction 'Ident' .aIdent oEmitIdent @FormalParameters
954         [=
955             | 'returns':
956                 .aReturns 'Ident' .aIdent oEmitIdent ':' @TypeDefinition
957             | *:
958         ]
959         '=' '<NL>'
960         @RoutineDefinition;
961
962
963
964     FormalParameters :
965
966         [=
967             | '(':
968                 .aParms @RoutineFormal
969                     {[=
970                         | ')':
971                             .aEndParms >
972                         | ',':
973                             @RoutineFormal
974                     ]}
975             | *:
976         ];
977
978
979
980     RoutineFormal :
981
982         @Pervasive @BindingCondition 'Ident' .aIdent oEmitIdent
983         {[=
984             | ',':
985                 'Ident' .aIdent oEmitIdent
986             | *:
987                 >
988         ]}
```

```
989         ':' .aParmType @TypeDefinition;
990
991
992
993     RoutineDefinition :
994
995         oEmitLine
996         @ImportClauses
997         [=
998             | 'pre':
999                 [=
000                     | '(':
001                         .aPre @Expression ')' ';'
002                         oEmitLine
003                     | *:
004                         ';'
005                 ]
006             | *:
007         ]
008         [=
009             | 'post':
010                 [=
011                     | '(':
012                         .aPost @Expression ')' ';'
013                         oEmitLine
014                     | *:
015                         ';'
016                 ]
017             | *:
018         ]
019         @RoutineBody;
020
021
022
023     RoutineBody :
024
025         [=
026             | 'begin':
027                 .aBegin
028                 @BlockBody
029                 'end' .aEndBegin
030                 [=
031                     | 'Ident':
032                         .aEndIdent .aIdent oEmitIdent
033                     | *:
034                 ]
035             | 'code':
036                 .aCode oEmitLine
037                 [=
038                     | 'CodeStart':
039                         {
040                             [=
```

```
041                                 | 'CodeMiddle':
042                                     .aCodeUnit oEmitCode
043                                 | *:
044                                     'CodeEnd' >
045                             ]
046                             oEmitLine
047                         }
048                     | *:
049                 ]
050                 'end' .aEndCode
051                 [=
052                     | 'Ident':
053                         .aEndIdent .aIdent oEmitIdent
054                     | *:
055                 ]
056             | 'forward':
057                 .aForward
058             | 'external':
059                 .aExternal
060         ];
061
062
063     %     Statements
064
065     Statements :
066
067         {
068             [=
069                 | 'Ident':
070                     @AssignmentOrCallStatement
071                 | 'exit':
072                     @ExitStatement
073                 | 'return':
074                     @ReturnStatement
075                 | 'assert':
076                     @AssertStatement
077                 | 'begin':
078                     @BeginStatement
079                 | 'if':
080                     @IfStatement
081                 | 'loop':
082                     @LoopStatement
083                 | 'for':
084                     @ForLoopStatement
085                 | 'case':
086                     @CaseStatement
087                 | ';':
088                 | *:
089                     ';' >
090             ]
091             ';' oEmitLine
092         };
```

```
093
094
095
096     AssignmentOrCallStatement :
097
098         @Variable
099         [=
100             | ':=':
101                 .aAssign @Expression
102             | *:
103         ];
104
105
106
107     ExitStatement :
108
109         .aExit
110         [=
111             | 'when':
112                 .aWhen @Expression
113             | *:
114         ];
115
116
117
118     ReturnStatement :
119
120         .aReturn
121         [=
122             | '(':
123                 .aReturnValue @Expression ')'
124             | *:
125         ]
126         [=
127             | 'when':
128                 .aWhen @Expression
129             | *:
130         ];
131
132
133
134     AssertStatement :
135
136         [=
137             | '(':
138                 .aAssert @Expression ')'
139             | *:
140         ];
141
142
143
144     BeginStatement :
```

```
145
146         .aBegin
147         @BlockBody
148         'end' .aEndBegin;
149
150
151
152     BlockBody :
153
154         oEmitLine
155         [=
156             | 'not':
157                 'checked' .aNotChecked ';'
158                 oEmitLine
159             | 'checked':
160                 .aChecked ';'
161                 oEmitLine
162             | *:
163         ]
164         @Declarations
165         @Statements;
166
167
168
169     IfStatement :
170
171         .aIf @Expression 'then' '<NL>'
172         @BlockBody
173         @ElseClause
174         'end' 'if' .aEndIf;
175
176
177
178     ElseClause :
179
180         [=
181             | 'elseif':
182                 .aElse oEmitLine
183                 .aIf @Expression 'then' '<NL>'
184                 @BlockBody
185                 @ElseClause
186                 .aEndIf oEmitLine
187             | 'else':
188                 .aElse '<NL>'
189                 @BlockBody
190             | *:
191         ];
192
193
194
195     LoopStatement :
196
```

```
197         .aLoop '<NL>'
198         @BlockBody
199         'end' 'loop' .aEndLoop;
200
201
202
203     ForLoopStatement :
204
205         .aFor 'Ident' .aIdent oEmitIdent
206         [=
207             | 'decreasing':
208                 .aDecreasing
209             | *:
210         ]
211         'in' .aIn @Expression
212         [=
213             | '..':
214                 .aTo @Sum
215             | *:
216         ]
217         oEmitLine
218         'loop' @LoopStatement;
219
220
221
222     CaseStatement :
223
224         .aCase
225         [=
226             | 'var':
227                 .aVar 'Ident' .aIdent oEmitIdent 'bound' 'to' .aBound
228                 'Ident' @Variable
229             | 'readonly':
230                 .aReadonly 'Ident' .aIdent oEmitIdent 'bound' 'to' .aBound
231                 'Ident' @Variable
232             | 'const':
233                 .aConst 'Ident' .aIdent oEmitIdent ':=' .aAssign 'Ident'
234                     @Variable
235             | 'Ident':
236                 [=
237                     | ':=':
238                         .aConst .aIdent oEmitIdent .aAssign 'Ident'
239                             @Variable
240                     | 'bound':
241                         'to' .aReadonly .aIdent oEmitIdent .aBound
242                         'Ident' @Variable
243                     | *:
244                         @Variable @PartialSum
245                 ]
246             | *:
247                 @Sum
248         ]
```

```
249         'of' '<NL>'
250         @CaseBody .aEndCase;
251
252
253     CaseBody :
254
255         oEmitLine
256         {[=
257             | 'end':
258                 'case' >
259             | 'otherwise':
260                 '=>' '<NL>'
261                 .aOtherwise @BlockBody
262                 'end' 'case' >
263             | *:
264                 .aLabels @CaseLabel
265                 {[=
266                     | ',':
267                         @CaseLabel
268                     | *:
269                         >
270                 ]}
271                 .aEndLabels '=>' '<NL>'
272                 @BlockBody
273                 'end' .aEndLabel @CaseLabel ';'
274                 oEmitLine
275         ]};
276
277
278     CaseLabel :
279
280         @Expression
281         [=
282             | '..':
283                 .aTo @Sum
284             | *:
285         ];
286
287
288     %    Expressions
289
290     Expression :
291
292         @SubExpression .aEndExpression;
293
294
295     SubExpression :
296
297         @Disjunction
298
299
300
```

```
301         [=
302             | '->':
303                 .aInfixImply @Disjunction .aImply
304             | *:
305         ];
306
307
308
309     Disjunction :
310
311         @Conjunction
312         {[=
313             | 'or':
314                 .aInfixOr @Conjunction .aOr
315             | *:
316                 >
317         ]};
318
319
320
321     Conjunction :
322
323         @Negation
324         {[=
325             | 'and':
326                 .aInfixAnd @Negation .aAnd
327             | *:
328                 >
329         ]};
330
331
332
333     Negation :
334
335         [=
336             | 'not':
337                 @Relation .aNot
338             | *:
339                 @Relation
340         ];
341
342
343
344     Relation :
345
346         @SubSum
347         [=
348             | '=':
349                 .aInfixCompare @SubSum .aEqual
350             | '<':
351                 .aInfixCompare @SubSum .aLess
352             | '<=':
```

```
353                 .aInfixCompare @SubSum .aLessEqual
354             | '>':
355                 .aInfixCompare @SubSum .aGreater
356             | '>=':
357                 .aInfixCompare @SubSum .aGreaterEqual
358             | 'not':
359                 [=
360                     | '=':
361                         .aInfixCompare @SubSum .aNotEqual
362                     | 'in':
363                         .aNotIn @Sum
364                         [=
365                             | '..':
366                                 .aTo @Sum
367                             | *:
368                         ]
369                 ]
370             | 'in':
371                 .aIn @Sum
372                 [=
373                     | '..':
374                         .aTo @Sum
375                     | *:
376                 ]
377             | *:
378         ];
379
380
381     Sum :
382
383         @SubSum .aEndExpression;
384
385
386     SubSum :
387
388         @Factor @PartialSubSum;
389
390
391     PartialSum :
392
393         @PartialSubSum .aEndExpression;
394
395
396     PartialSubSum :
397
398         @PartialTerm
399         {[=
400             | '+':
```

```
405                 @Term .aAdd
406             | '-':
407                 @Term .aSubtract
408             | 'xor':
409                 @Term .aXor
410             | *:
411                 >
412         ]};
413
414
415
416     Term :
417
418         @Factor @PartialTerm;
419
420
421
422     PartialTerm :
423
424         {[=
425             | '*':
426                 @Factor .aMultiply
427             | 'div':
428                 @Factor .aDiv
429             | 'mod':
430                 @Factor .aMod
431             | *:
432                 >
433         ]};
434
435
436
437     Factor :
438
439         [=
440             | 'Ident':
441                 @Variable
442             | 'Number', 'OctalNumber', 'HexNumber':
443                 .aNumber oEmitNumber
444             | 'Char', 'ExChar':
445                 .aChar oEmitChar
446             | 'MDChar':
447                 .aMDChar oEmitChar
448             | 'String', 'ExString':
449                 .aString oEmitString
450             | 'MDString':
451                 .aMDString oEmitString
452             | '(':
453                 @SubExpression ')' .aParens
454             | '-':
455                 @Factor .aMinus
456         ];
```

```
457
458
459
460     Variable :
461
462         .aIdent oEmitIdent
463         {[=
464             | '(':
465                 .aSubs @Subscripts .aEndSubs
466             | '.':
467                 .aField 'Ident' .aIdent oEmitIdent
468             | '^':
469                 .aPointer
470             | *:
471                 >
472         ]};
473
474
475
476     Subscripts :
477
478         {
479             [=
480                 | ')':
481                     >
482                 | 'parameter':
483                     .aParameter
484                 | 'any':
485                     .aAny
486                 | 'unknown':
487                     .aUnknown
488                 | 'all':
489                     .aAll
490                 | *:
491                     @Expression
492                     [=
493                         | '..':
494                             .aTo @Sum
495                         | *:
496                     ]
497             ]
498             [=
499                 | ')':
500                     >
501                 | ',':
502             ]
503         };
504
505
506
507     _|_
```

```
include 'IOSTART'
include 'POWERSET'

var Assembler:
    module

        imports (Abort, var IO);


        {   Project Euclid

            Module: Syntax/Semantic Language Assembler V4.23
            Author: J.R. Cordy
            Reader: D.R. Crowe
            Date:   1 October 1979                                  }


        { Copyright (C) 1977,1978,1979 The University of Toronto }


        {   This module assembles the Syntax/Semantic Language
         into tables to be interpreted by the S/SL Table Walker
         program.  The Parser pass and Semantic passes of the
         Euclid compiler each will be implemented as Syntax/
         Semantic Language programs.

            The Assembler accepts as input a Syntax/Semantic
         Language program as described in Project Euclid
         working paper 5.  Its output is Euclid code to define
         the constants and symbols used in the program
         and the declaration of the initialized Syntax/Semantic
         table.

            For detailed information on the Syntax/Semantic
         Langauge see Euclid working paper 5.  For information
         on the format of the assembled table see Euclid
         working paper 13.


         Automatic Generation of Syntax Error
             Recovery
         Grace Evans, Summer 1982  }


        { Predefined Character Constants }

        pervasive const blank := $$S;
        pervasive const tab := $$S; {rmb}
        pervasive const newLine := $$N;
        pervasive const endOfFile := $$E;
        pervasive const formFeed := $$F;
        pervasive const quote := $$';
```

```
53          pervasive const dollar := $$$;
54          pervasive const breakChar := $_;
55
56
57          { Syntax/Semantic Language Tokens }
58
59          type TokenType = Char;
60
61          pervasive const tName := $a;  { Identifiers }
62          pervasive const tString := quote;  { 'Quoted strings' }
63          pervasive const tNumber := $1;  { Positive integers }
64          pervasive const tColon  := $:;
65          pervasive const tSemicolon := $; ;
66          pervasive const tEqual  := $=;
67          pervasive const tMinus  := $-;
68          pervasive const tGoalpost := $_;  { _|_ }
69          pervasive const tInput  := $=;
70          pervasive const tOutput := $.;
71          pervasive const tError  := $#;
72          pervasive const tCall   := $@;
73          pervasive const tExit   := $>;
74          pervasive const tReturn := $<;  { >> }
75          pervasive const tLeftParen := $(;
76          pervasive const tRightParen := $);
77          pervasive const tCycle  := ${;
78          pervasive const tCycleEnd := $};
79          pervasive const tChoice := $[;
80          pervasive const tChoiceEnd := $];
81          pervasive const tComma  := $, ;
82          pervasive const tOr := $|;
83          pervasive const tAlternateOr := $!;
84          pervasive const tOtherwise := $*;
85          pervasive const tEndOfFile := endOfFile;
86
87
88          { Next Input Token }
89
90          var token:
91              TokenType;
92
93          pervasive const maxTokenLength := 50;
94          pervasive const maxNameLength := maxTokenLength;
95          pervasive const maxStringLength := maxNameLength;
96          pervasive const maxNumberLength := 5;
97
98          var tokenLength:
99              0 .. maxTokenLength;
100         var tokenText:
101             array 1 .. maxTokenLength of Char;
102
103
104         { Input channel }
```

```
105
106             const inputFile := 1;
107
108             { Output Channel }
109
110             const outputFile := 3;
111
112
113
114             { Next input character }
115
116             var nextChar:
117                 Char;
118
119
120             { Letter Normalization Map }
121
122             pervasive const ordCharFirst := 0;
123             pervasive const ordCharLast := 255;
124
125             var lowerCase:
126                 array ordCharFirst .. ordCharLast of Char;
127
128
129             { Assembly Error Codes }
130
131             pervasive type ErrorCodes =
132                 (eNoError, eSyntaxError, eIllegalClass, eDoubleDefinition,
133                  eWrongNameClass, eIllegalString, eMissingCycleEnd,
134                  eMissingChoiceEnd, eUndefinedProcedure, eNameTooLong,
135                  eStringTooLong, ePrematureEndOfFile, eUndefinedSymbol,
136                  eTooManyNameChars, eTooManyNames, eTableTooLarge,
137                  eCyclesTooDeep, eTooManyExits, eChoicesTooDeep,
138                  eTooManyLabels, eTooManyMerges, eTooManyCalls,
139                  eProcedureTooLarge, eTooManySets, eSequencesTooDeep,
140                  eHashTableTooFull, eNumberTooLong);
141
142             pervasive const firstFatalError := eTooManyNameChars;
143
144             var errors:
145                 Boolean := false;
146
147
148             { Current Line Number }
149
150             pervasive const maxLines := 30000;
151             var lineNo:
152                 0 .. maxLines := 0;
153
154
155             { Listing Control }
156
```

```
157             var listing:
158                 Boolean := false;
159
160             { The SetTable: used to save key sets of reachable symbols
161              and handles for error recovery }
162
163             const totalBits := 108;
164             const maxBits := 15;
165             const setSize := 7;
166
167             { Predefined flag values for PowerSet module }
168
169             const flagValue:
170                 array 0 .. maxBits of SignedInt :=
171                     (1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048,
172                      4096, 8192, 16384, 1 { unused } );
173
174             type SetType =
175                 array 0 .. setSize of Powerset;
176
177             type SetTableEntry =
178                 record
179                     var sett:
180                         SetType;  { set of 120 bits }
181                     var visiting:
182                         Boolean;  { Used to merge lists of setLabels into
183                                     pure 120-bit sets: indicates set is
184                                     being processed }
185                     var duplicate:
186                         Boolean;  { Indicates if set is duplicate of
187                                      another set  already in S-code table }
188                     var root:
189                         SignedInt;  { Index to linkTable: root of list of
190                                       set labels; root = 0 indicates empty
191                                       list; after elimination of duplicate
192                                       sets, root points to set location in
193                                       S-code table }
194                 end SetTableEntry;
195
196             const maxSets := 600;
197             var setTop:
198                 0 .. maxSets := 0;
199             var setTable:
200                 array 1 .. maxSets of SetTableEntry;
201             var handleSet:
202                 0 .. maxSets;  { Recovery tokens for S/SL procedure
203                                  currently being processed }
204
205             const maxLinks := 3000;
206
207             type Node =
208                 record
```

```
209                 var info:
210                     1 .. maxSets;  { SetLabel (index) of set in
211                                      setTable }
212                 var next:
213                     SignedInt;  { Index to linkTables: points to next
214                                   setLabel in list; next = 0 indicates
215                                   end of list }
216             end Node;
217
218         var linkTable:
219             array 1 .. maxLinks of Node;
220             { LinkTable: holds list of (children) to be merged with
221               parent set }
222         var linkTop:
223             SignedInt := 0;
224
225         { The Sequence Stack: used to save setLabels (indices) of sets
226           in setTable  which are in current sequence }
227
228         const maxSequences := 8;
229         var sequenceTop:
230             0 .. maxSequences := 0;
231
232         type SequenceStackEntry =
233             record
234                 var root:
235                     SignedInt;  { Index to sequenceLinkTable }
236                 var exit:
237                     Boolean;  { Handle and reachable set generation:
238                                 true indicates exit symbol at end of  }
239                                 sequence }
240                 var exposed:
241                     Boolean;  { HandleSet generation: true indicates
242                                 no recovery tokens have been added to
243                                 handleSet in sequence path }
244             end SequenceStackEntry;
245
246         var sequenceStack:
247             array 1 .. maxSequences of SequenceStackEntry;
248         var sequenceLinks:
249             array 1 .. maxLinks of Node;  { SequenceLinkTable: holds
250                                             list of setLabels in a
251                                             sequence }
252         var seqLinkTop:
253             SignedInt := 0;
254
255
256         { The HashTable: used to eliminate duplicate sets in the
257           setTable; contains pointer to setTable }
258
259         const hashSize := 599;
260         const tableFull := 569;  { 95% full }
```

```
261
262         var hashTable:
263             array 0 .. hashSize of 0 .. maxSets;
264
265         var slotsFilled:
266             0 .. hashSize := 0;
267
268
269
270
271         { Name Chars Storage }
272
273         const maxNameChars := 15000;
274         pervasive const none := 0;
275
276         var nameCharsTop:
277             0 .. maxNameChars := 0;
278         var nameChars:
279             array 0 .. maxNameChars of Char;
280
281
282         { Classes of Names }
283
284         pervasive type NameClasses =
285             (cInput, cOutput, cError, cSimpleOp, cParmOp, cChoiceOp,
286              cEmitOp, cValue, cProcedure, cClass);
287
288
289         { The Name Table:  used to save information about defined
290           symbols }
291
292         pervasive const nullValue := -9999;
293
294         type NameTableEntry =
295             record
296                 var class:
297                     NameClasses;
298                 var charsIndex:
299                     0 .. maxNameChars;
300                 var setLabel:
301                     0 .. maxSets;  { Location in setTable of handle
302                                      set of recovery tokens for S/Sl  }
303                                      procedures }
304                 var derivedEmpty:
305                     Boolean;  { HandleSet generation: true indicates S/SL
306                                 procedure can derive empty }
307                 var length:
308                     1 .. maxNameLength;
309                 var value:
310                     SignedInt;
311             end NameTableEntry;
312
```

```
313             const maxNames := 1000;
314             pervasive const notFound := 0;
315
316             var nameTop:
317                 0 .. maxNames := 0;
318             var nameTable:
319                 array 0 .. maxNames of NameTableEntry;
320
321             { nameIndex is the index in the Name Table of the name last
322              referenced; nameIndex = 0 (notFound) indicates the referenced
323              name is not present in the Name Table }
324
325             var nameIndex:
326                 0 .. maxNames;
327
328
329             { Value signs }
330
331             type Signs = (positive, negative);
332
333
334             { Next Name Value }
335
336             var nextValue:
337                 array cInput .. cValue of SignedInt;
338
339
340             { Current Definition Class }
341
342             var definitionClass:
343                 NameClasses := cInput;
344
345
346             { Predefined Name Table Entries }
347
348             const noPredefinedNames := 8;
349
350             const predefinedNameClass:
351                 array 0 .. noPredefinedNames of NameClasses :=
352                     (cInput { Unused }, cInput, cOutput, cError,
353                      cSimpleOp, ParmOp, cChoiceOp, cEmitOp, cValue);
354
355             const noPredefinedNameChars := 57;
356
357             const predefinedNameChars:
358                 array 0 .. noPredefinedNameChars of Char :=
359                     (blank { Unused },
360                      $i, $n, $p, $u, $t, $s,
361                      $o, $u, $t, $p, $u, $t, $s,
362                      $e, $r, $r, $o, $r, $s,
363                      $s, $i, $m, $p, $l, $e, $o, $p, $s,
364                      $p, $a, $r, $m, $o, $p, $s,
```

```
365                 $c, $h, $o, $i, $c, $e, $o, $p, $s,
366                 $e, $m, $i, $t, $o, $p, $s,
367                 $v, $a, $l, $u, $e, $s);
368         const predefinedNameLength:
369             array 0 .. noPredefinedNames of SignedInt :=
370                 (1 { Unused }, 6, 7, 6, 9, 7, 9, 7, 6);
371
372
373         { Primitive Table Operations }
374
375         pervasive const firstPrimitiveOperation := 0;
376         pervasive const oCall := 0;
377         pervasive const oReturn := 1;
378         pervasive const oRepeat := 2;
379         pervasive const oMerge := 3;
380         pervasive const oInput := 4;
381         pervasive const oInputChoice := 5;
382         pervasive const oEmit := 6;
383         pervasive const oError := 7;
384         pervasive const oChoiceEnd := 8;
385         pervasive const lastPrimitiveOperation := 8;
386
387
388         { Number of error codes reserved for Table Walker system use }
389
390         const noReservedErrorCodes := 10;
391
392
393         { The Assembled Table }
394
395         pervasive const maxTableSize := 30000;
396         pervasive const nullAddress := 0;
397
398         var tableAddress:
399             0 .. maxTableSize := 0;
400         var saveTableAddress:
401             0 .. maxTableSize;
402
403         const tableFile := 2;    { I/O channel for assembled table }
404
405
406         { The Table Fragment for the Current Procedure }
407
408         pervasive const maxProcedureSize := 1000;
409
410         var procedureBase:
411             0 .. maxTableSize := 0;      { Start address of current
412                                            procedure }
413         var table:
414             array 0 .. maxProcedureSize of SignedInt;
415
416
```

```
417             { The Call Table:  used to save the table addresses of
418               calls and the Name Table index of the called procedure }
419
420             const maxCalls := 900;
421
422             type CallTableEntry =
423                 record
424                     var callAddress:
425                         0 .. maxTableSize;
426                     var procedureIndex:
427                         0 .. maxNames;
428                 end CallTableEntry;
429
430             var callTop:
431                 0 .. maxCalls;
432             var callTable:
433                 array 1 .. maxCalls of CallTableEntry;
434
435
436             { Cycle Handling }
437
438             { The Exit Stack:  used to save the table addresses of cycle
439               exits }
440
441             const maxExits := 15;
442
443             var exitTop:
444                 0 .. maxExits := 0;
445             var exitAddress:
446                 array 1 .. maxExits of 0 .. maxTableSize;
447
448             { The Cycle Stack:  used in handling cycle constructs }
449
450             type CycleStackEntry  =
451                 record
452                     var cycleAddress:
453                         0 .. maxTableSize;
454                     { exitIndex is the origin of the portion of the Exit
455                      Stack for this cycle }
456                     var exitIndex:
457                         0 .. maxExits;
458                     var cycleSet:
459                         0 .. maxSets;  { Error recovery tokens }
460                     var initExposed:
461                         Boolean;  { Initial cycle condition: true
462                                     indicates no tokens added to
463                                     handleSet }
464                     var finalExposed:
465                         Boolean;  { Final cycle condition: true if default
466                                     condition (*: >) at inputChoice }
467                 end CycleStackEntry;
468
```

```
469             const maxCycles := 7; { Deep }
470
471             var cycleTop:
472                 0 .. maxCycles := 0;
473             var cycleStack:
474                 array 1 .. maxCycles of CycleStackEntry;
475
476
477             { Choice Handling }
478
479             { The Merge Stack:  used to save the addresses of the merge
480              branches following each alternative of a choice }
481
482             const maxMerges := 63;
483
484             var mergeTop:
485                 0 .. maxMerges := 0;
486             var mergeAddress:
487                 array 1 .. maxMerges of 0 .. maxTableSize;
488
489             { The Label Stack:  used to save the alternative values and
490               corresponding table addresses in a choice }
491
492             const maxLabels := 63;
493
494             var labelTop:
495                 0 .. maxLabels := 0;
496             var labelValue:
497                 array 1 .. maxLabels of SignedInt;
498             var labelAddress:
499                 array 1 .. maxLabels of 0 .. maxTableSize;
500
501             { The Choice Stack }
502
503             type ChoiceStackEntry  =
504                 record
505                     var choiceClass:
506                         NameClasses;
507                     var choiceAddress:
508                         0 .. maxTableSize;
509                     { mergeIndex is the origin of the portion of the
510                      Merge Stack for this choice }
511                     var mergeIndex:
512                         0 .. maxMerges;
513                     { labelIndex is the origin of the portion of the
514                      Lable Stack for this choice }
515                     var labelIndex:
516                         0 .. maxLabels;
517                     var otherwisePresent:
518                         Boolean;
519                     var firstAlternative:
520                         Boolean;
```

```
521                 var root:
522                     SignedInt;  { Temporarily points to choice
523                                   alternatives }
524                 var initExposed:
525                     Boolean;  { HandleSet generation: initial
526                                 condition upon entering choice }
527                 var finalExposed:
528                     Boolean;  { Condition at end of all choice
529                                 alternatives }
530             end ChoiceStackEntry;
531
532         const maxChoices := 7; { Deep }
533
534         var choiceTop:
535             0 .. maxChoices := 0;
536         var choiceStack:
537             array 1 .. maxChoices of ChoiceStackEntry;
538
539         { For printing table }
540
541         const numLength := 5;
542         type NumberArray =
543              array 1 .. numLength of Char;
544
545         { The primitive opCode table: holds strings for printing }
546
547         const codeLength := 11;
548
549         type AssemCode =
550             array 1 .. codeLength of Char;
551
552         const numberOfSyntaxOps := 8;
553
554         var opCodeTable:
555             array 0 .. numberOfSyntaxOps of AssemCode;
556
557         { The choice Table: indicates nameTable index where
558           inputChoice Table begins }
559
560         const choices := 20;
561
562         var choiceTable:
563             array 0 .. choices of signedInt;
564         var inputChoiceTableTop:
565             0 .. choices := 0;
566
567         { For printing literal strings }
568
569         const stringLength := 15;
570
571         type stringVarying =
572             packed array 1 .. 32767 of Char;
```

```
573         const blanks := '                    ';
574
575         procedure ConvertNumber (num: SignedInt) =
576
577             imports (var IO, numLength, NumberArray,outputFile);
578
579             { Converts integer to char, stores, and emits for table }
580
581             begin
582                 var number:
583                     NumberArray;
584                 var i:
585                     SignedInt;
586                 var j:
587                     SignedInt;
588
589                 i := num;
590                 if i = 0 then
591                     number := '    0';
592                 elseif j > 0 then
593                     number := '     ';
594                     j := numLength;
595
596                     { Convert integer to character }
597
598                     loop
599                         exit when i <= 0;
600                         number (j) := chr (i mod 10 + ord ('0'));
601                         i := i div 10;
602                         j := j - 1;
603                     end loop;
604
605                 else  { Negative number }
606                     number := 'error';
607                 end if;
608
609                 { Emit characters }
610
611                 j := 1;
612                 loop
613                     exit when j > numLength;
614                     IO.WriteChar (outputFile, number (j));
615                     j := j + 1;
616                 end loop;
617
618             end ConvertNumber;
619
620         procedure WriteLine =
621
622             imports (var IO, outputFile);
623
624             { Terminates a print line }
```

```
625
626             begin
627                 IO.WriteChar (outputFile, $$N);
628             end WriteLine;
629
630
631         procedure WriteString (outputString: stringVarying) =
632
633             imports (var IO);
634
635             { Write string to outputFile }
636
637             begin not checked
638                 var k:
639                     SignedInt;
640
641                 k := 1;
642
643                 loop
644                     exit when outputString (k) = $$E;
645                     IO.WriteChar (outputFile, outputString (k));
646                     k := k + 1;
647                 end loop;
648
649             end WriteString;
650
651
652
653
654
655         procedure PutName =
656
657             imports (var IO, nameTable, nameIndex, nameChars, nameTop,
658                      WriteString, WriteLine, outputFile);
659
660             pre (nameIndex <= nameTop);
661
662             begin
663                 var i:
664                     1 .. maxNameLength;
665
666                 IO.WriteChar (outputFile, $");
667
668                 i := 1;
669                 loop
670                     IO.WriteChar (outputFile, nameChars
671                                   (nameTable(nameindex).charsIndex
672                                                        + i - 1));
673                     exit when i = nameTable(nameIndex).length;
674                     i := i + 1;
675                 end loop;
676
```

```
677                 IO.WriteChar (outputFile, $");
678
679             end PutName;
680
681
682         procedure Error (errorNo: ErrorCodes) =
683
684             imports (var IO, Abort, var errors, lineNo, PutName,
685                      WriteLine, WriteString, ConvertNumber);
686
687             pre (errorNo not = eNoError);
688
689             { This procedure Emits the error message associated with
690               errorNo }
691
692             begin
693                 WriteLine;
694                 WriteString ('*** Line ');
695                 ConvertNumber (lineNo);
696                 WriteString (': ');
697
698                 case errorNo of
699
700                     eSyntaxError =>
701                         WriteString ('Syntax Error');
702                         end eSyntaxError;
703
704                     ePrematureEndOfFile =>
705                         WriteString ('Unexpected End Of File');
706                         end ePrematureEndOfFile;
707
708                     eNameTooLong =>
709                         WriteString ('Symbol Too Long');
710                         end eNameTooLong;
711
712                     eNumberTooLong =>
713                         WriteString ('Value Too Large');
714                         end eNumberTooLong;
715
716                     eStringTooLong =>
717                         WriteString ('String Too Long');
718                         end eNumberTooLong;
719
720                     eIllegalClass =>
721                         WriteString ('Illegal Class Name');
722                         end eIllegalClass;
723
724                     eUndefinedSymbol =>
725                         WriteString ('Symbol ');
726                         PutName;
727                         WriteString (' Undefined');
728                         end eUndefinedSymbol;
```

```
729
730                     eDoubleDefinition =>
731                         WriteString ('Symbol ');
732                         PutName;
733                         WriteString (' Previously Defined');
734                         end eDoubleDefinition;
735
736                     eWrongNameClass =>
737                         WriteString ('Illegal Context for Symbol ');
738                         PutName;
739                         end eWrongNameClass;
740
741                     eUndefinedProcedure =>
742                         WriteString ('Procedure ');
743                         PutName;
744                         WriteString (' Undefined');
745                         end eUndefinedProcedure;
746
747                     eIllegalString =>
748                         WriteString ('Illegal String Synonym');
749                         end eIllegalString;
750
751                     eMissingCycleEnd =>
752                         WriteString ('Missing Cycle End "}"');
753                         end eMissingCycleEnd;
754
755                     eMissingChoiceEnd =>
756                         WriteString ('Missing Choice End "]"');
757                         end eMissingChoiceEnd;
758
759                     eTooManyNameChars =>
760                         WriteString ('Too Many Symbols (Chars)');
761                         end eTooManyNameChars;
762
763                     eTooManyNames =>
764                         WriteString ('Too Many Symbols');
765                         end eTooManyNames;
766
767                     eTableTooLarge =>
768                         WriteString ('Table Too Large');
769                         end eTableTooLarge;
770
771                     eProcedureTooLarge =>
772                         WriteString ('Procedure Too Large');
773                         end eProcedureTooLarge;
774
775                     eTooManyCalls =>
776                         WriteString ('Too Many Procedure Calls');
777                         end eTooManyCalls;
778
779                     eCyclesTooDeep =>
780                         WriteString ('Cycles Too Deep');
```

```
781                         end eCyclesTooDeep;
782
783                     eChoicesTooDeep =>
784                         WriteString ('Choices Too Deep');
785                         end eChoicesTooDeep;
786
787                     eSequencesTooDeep =>
788                         WriteString ('Sequences Too Deep');
789                         end eSequencesTooDeep;
790
791                     eTooManySets =>
792                         WriteString ('Too Many Sets');
793                         end eTooManySets;
794
795                     eTooManyExits =>
796                         WriteString ('Too Many Cycle Exits');
797                         end eTooManyExits;
798
799                     eTooManyLabels, eTooManyMerges =>
800                         WriteString ('Too Many Alternatives');
801                         end eTooManyLabels {, eTooManyMerges };
802
803                     eHashTableTooFull =>
804                         WriteString ('Hash Table Almost Full');
805                         end eHashTableTooFull;
806
807                 end case;
808
809                 WriteLine;
810                 errors := true;
811
812                 if errorNo >= firstFatalError then
813                     WriteLine;
814                     WriteString ('*** Assembly Aborted');
815                     WriteLine;
816                     Abort;
817                 end if;
818
819             end Error;
820
821
822         procedure ReadNextChar =
823
824             imports (var IO, inputFile, var nextChar, tableAddress,
825                      ConvertNumber, WriteLine, WriteString, listing,
826                      var lineNo, outputFile);
827
828             { Gets and prints next input character }
829
830             begin
831                 if listing and lineNo not = 0 then
832                     IO.WriteChar (outputFile, nextChar);
```

```
833                 end if;
834
835                 if nextChar = newLine then
836                     if lineNo < maxLines then
837                         lineNo := lineNo + 1;
838                     else
839                         lineNo := 0;
840                     end if;
841
842                     if listing then
843                         ConvertNumber (tableAddress);
844                         IO.WriteChar (outputFile, tab);
845                     end if;
846                 end if;
847
848                 if nextChar not = endOfFile then
849                     IO.ReadChar (inputFile, nextChar);
850
851                     if IO.EndFile (inputFile) then
852                         nextChar := endOfFile;
853                     end if;
854                 end if;
855
856             end ReadNextChar;
857
858
859         procedure GetToken =
860
861             imports  (var token, var tokenText,  var tokenLength,
862                 var nextChar, Error, ReadNextChar);
863
864            pre (maxTokenLength >= maxNameLength and
865                 maxTokenLength >= maxStringLength and
866                 maxTokenLength >= maxNumberLength);
867
868             post { Token, tokenText and tokenLength have appropriate
869                    values };
870
871             begin
872                 var errorCode:
873                     ErrorCodes;
874
875                 errorCode := eNoError;
876
877                 { Skip blanks, newLines and comments }
878
879                 loop
880                     exit when nextChar not = blank
881                         and nextChar not = newLine
882                         and nextChar not = tab
883                         and nextChar not = formFeed;
884                     ReadNextChar;
```

```
885
886                     if nextChar = $% then
887                         { Skip comment }
888                         loop
889                             ReadNextChar;
890                             exit when nextChar = newLine or
891                                       nextChar = endOfFile;
892                         end loop;
893                     end if;
894                 end loop;
895
896                 { Scan and set token }
897
898                 tokenLength := 0;
899
900                 if (nextChar >= $a and nextChar <= $z) or
901                         (nextChar >= $A and nextChar <= $Z) then
902
903                     { Scan identifier }
904                     loop
905                         if tokenLength < maxNameLength then
906                             tokenLength := tokenLength + 1;
907                             tokenText(tokenLength) := nextChar;
908                         else
909                             errorCode := eNameTooLong;
910                         end if;
911
912                         ReadNextChar;
913                         exit when not ((nextChar >= $a and
914                                         nextChar <= $z) or
915                             (nextChar >= $A and nextChar <= $Z) or
916                             (nextChar >= $0 and nextChar <= $9) or
917                             nextChar = breakChar);
918                     end loop;
919
920                     token := tName;
921
922                     { Handle keyword alternates }
923                     if tokenLength = 2 then
924
925                         if tokenText(1) = $i and
926                                 tokenText(2) = $f then
927                             token := tChoice;
928
929                         elseif tokenText(1) = $f and
930                                 tokenText(2) = $i then
931                             token := tChoiceEnd;
932
933                         elseif tokenText(1) = $d and
934                                 tokenText(2) = $o then
935                             token := tCycle;
936
```

```
937                         elseif tokenText(1) = $o and
938                                 tokenText(2) = $d then
939                             token := tCycleEnd;
940                         end if;
941                     end if;
942
943                 elseif nextChar >= $0 and nextChar <= $9 then
944
945                     { Scan number }
946                     loop
947                         if tokenLength < maxNumberLength then
948                             tokenLength := tokenLength + 1;
949                             tokenText(tokenLength) := nextChar;
950                         else
951                             errorCode := eNumberTooLong;
952                         end if;
953
954                         ReadNextChar;
955                         exit when nextChar < $0 or nextChar > $9;
956                     end loop;
957
958                     token := tNumber;
959
960                 elseif nextChar = quote then
961
962                     { Scan String }
963                     loop
964                         if tokenLength < maxStringLength-1 then
965                             tokenLength := tokenLength + 1;
966                             tokenText(tokenLength) := nextChar;
967                         else
968                             errorCode := eStringTooLong;
969                         end if;
970
971                         ReadNextChar;
972                         exit when nextChar = quote or
973                             nextChar = newLine or
974                             nextChar = endOfFile;
975                     end loop;
976
977                     tokenLength := tokenLength + 1;
978                     tokenText(tokenLength) := quote;
979
980                     if nextChar = quote then
981                         ReadNextChar;
982                     end if;
983
984                     token := tString;
985
986                 else
987                     { Special Symbols }
988                     token := nextChar;
```

```
989                     ReadNextChar;
990                     tokenLength := 1;
991                     tokenText(1) := token;
992
993                     if token = tExit and nextChar = $> then
994                         token := tReturn;
995                         ReadNextChar;
996
997                     elseif token = tGoalpost then
998                         ReadNextChar; { Flush |_ }
999                         ReadNextChar;
000
001                     elseif token = tAlternateOr then
002                         token := tOr;
003                     end if;
004                 end if;
005
006                 if errorCode not = eNoError then
007                     Error (errorCode);
008                 end if;
009
010             end GetToken;
011
012
013         procedure VerifyToken (expectedToken: TokenType) =
014
015             imports (var token, Error);
016
017             pre { Token is initialized };
018
019             post { Token = expectedToken or error message has been
020                    emitted };
021
022             begin
023                 if token not = expectedToken then
024                     Error (eSyntaxError);
025                     token := expectedToken;
026                     { Leave token text as is for future error
027                       messages }
028                 end if;
029
030             end VerifyToken;
031
032
033
034         procedure GetTokenAndVerify (expectedToken: TokenType) =
035
036             imports (VerifyToken, GetToken);
037
038             begin
039                 GetToken;
040                 VerifyToken (expectedToken);
```

```
041
042             end GetTokenAndVerify;
043
044
045         procedure EnterName (nameClass: NameClasses) =
046
047             imports (var nameTable, var nameTop, var nameIndex,
048                 var nameChars, var nameCharsTop, maxNameChars,
049                 maxNames, token, tokenText, tokenLength, Error,
050                 maxNameLength);
051
052             pre ((token = tName or token = tString) and tokenLength
053                   > 0);
054
055             post { Name (or string) has been entered in the nameTable
056                 if there is room;  otherwise, an error message has
057                 been emitted };
058
059             begin
060                 var i:
061                     0 .. maxNameLength;
062
063                 { Enter in nameTable }
064
065                 if nameTop < maxNames then
066                     nameTop := nameTop + 1;
067
068                     if nameCharsTop  +  tokenLength < maxNameChars
069                         then
070                         { Enter name chars in name chars table }
071                         nameTable(nameTop).charsIndex := nameCharsTop
072                                                              + 1;
073
074                         i := 1;
075                         loop
076                             nameCharsTop := nameCharsTop + 1;
077                             nameChars(nameCharsTop) := tokenText(i);
078                             exit when i = tokenLength;
079                             i := i + 1;
080                         end loop;
081
082                         nameTable(nameTop).length := tokenLength;
083                         nameTable(nameTop).class := nameClass;
084                         nameTable(nameTop).value := nullValue;
085
086                     else
087                         Error (eTooManyNameChars);
088                     end if;
089
090                 else
091                     Error (eTooManyNames);
092                 end if;
```

```
093
094                 nameIndex := nameTop;
095
096             end EnterName;
097
098
099         procedure LookupName =
100
101             imports (nameTable, nameChars, nameTop, var nameIndex,
102                 token, tokenLength, tokenText, lowerCase);
103
104             pre ((token = tName or token = tString) and
105                   tokenLength > 0);
106
107             post { nameIndex is the index in nameTable of the entry
108                 for the name if present and notFound otherwise };
109
110             { This procedure looks up a name (or string) in the
111               nameTable }
112
113             begin
114                 var i:
115                     1 .. maxNameLength;
116
117                 nameIndex := nameTop;
118                 loop
119                     exit when nameIndex = notFound;
120
121                     if nameTable(nameIndex).length = tokenLength then
122                         i := 1;
123                         loop
124                             exit when lowerCase (Char.Ord(nameChars
125                                 (nameTable (nameIndex).charsIndex
126                                                         + i - 1)))
127                                 not = lowerCase (Char.Ord
128                                           (tokenText(i))) or
129                                 i = tokenLength;
130                             i := i + 1;
131                         end loop;
132
133                         exit when lowerCase (Char.Ord(nameChars
134                             (nameTable(nameIndex).charsIndex
135                                                     + i - 1))) =
136                                 lowerCase (Char.Ord(tokenText(i)));
137                     end if;
138
139                     nameIndex := nameIndex  - 1;
140                 end loop;
141
142             end LookupName;
143
144
```

```
145         procedure VerifyNameClass (requiredClass: NameClasses) =
146
147             imports (nameTable, nameIndex, EnterName, Error);
148
149             pre { nameIndex has a value };
150
151             post { Error message has been emitted if name is not of the
152                 required class;  if undefined, name has been entered
153                 in the Name Table with the required class };
154
155             begin
156                 if nameIndex = notFound then
157                     EnterName (requiredClass);
158                     Error (eUndefinedSymbol);
159                 elseif nameTable(nameIndex).class not = requiredClass
160                     then
161                     Error (eWrongNameClass);
162                 end if;
163
164             end VerifyNameClass;
165
166
167
168         procedure LookupNameAndVerify (requiredClass: NameClasses) =
169
170             imports (LookupName, VerifyNameClass);
171
172             begin
173                 LookupName;
174                 VerifyNameClass (requiredClass);
175
176             end LookupNameAndVerify;
177
178
179         procedure VerifyAndEnterName =
180
181             imports (nameIndex, LookupName, EnterName, definitionClass,
182                 Error, token, tokenLength);
183
184             pre ((token = tName or token = tString) and
185                   tokenLength > 0);
186
187             post { Name (or string) has been verified and entered in
188                 the nameTable if there is room; otherwise, an error
189                 message has been emitted };
190
191             begin
192                 { Check for already defined }
193                 LookupName;
194
195                 if nameIndex not = notFound then
196                     Error (eDoubleDefinition);
```

```
197                 end if;
198
199                 { Enter in nameTable }
200                 EnterName (definitionClass);
201
202             end VerifyAndEnterName;
203
204
205
206         procedure EnterValue (nmIndex: 0..maxNames,
207                               nmValue: SignedInt) =
208
209             imports (var nameTable);
210
211             { Enters nmValue as value of nameTable(nmIndex) }
212
213             begin
214                 assert (nameTable(nmIndex).value = nullValue);
215                 nameTable(nmIndex).value := nmValue;
216
217             end EnterValue;
218
219
220         procedure VerifyAndSetClass =
221
222             imports (var definitionClass, LookupName, nameTable,
223                 nameIndex, predefinedNameClass, noPredefinedNames,
224                 nameChars, var nextValue, token, tokenLength, Error);
225
226             pre (token = tName and tokenLength > 0);
227
228             post { definitionClass is set to the class specified by
229                    the name };
230
231             begin
232                 var nextOperationValue:
233                     SignedInt;
234
235                 LookupName;
236
237                 { Keep operation codes unique }
238
239                 if definitionClass = cSimpleOp or
240                         definitionClass = cParmOp or
241                         definitionClass = cEmitOp or
242                         definitionClass = cChoiceOp then
243                     assert { For each operation class c, nextValue(c) <=
244                             <= nextValue(definitionClass) };
245                     nextOperationValue := nextValue(definitionClass);
246                     nextValue(cSimpleOp) := nextOperationValue;
247                     nextValue(cParmOp) := nextOperationValue;
248                     nextValue(cEmitOp) := nextOperationValue;
```

```
249                     nextValue(cChoiceOp) := nextOperationValue;
250                 end if;
251
252                 if nameIndex not = notFound and
253                         nameTable(nameIndex).class = cClass then
254                     assert (nameIndex <= noPredefinedNames);
255                     definitionClass := predefinedNameClass(nameIndex);
256                 else
257                     Error (eIllegalClass);
258                     definitionClass := cInput;
259                 end if;
260
261             end VerifyAndSetClass;
262
263
264         function EvaluateNumber (sign: Signs) returns result:
265                                                 SignedInt =
266
267             imports (token, tokenLength, tokenText);
268
269             pre (token = tNumber and tokenLength > 0 and
270                 tokenLength <= maxNumberLength);
271
272             post { Number is verified, evaluated and value returned };
273
274             begin
275                 var value:
276                     SignedInt;
277                 var i:
278                     1 .. maxNumberLength;
279
280                 value := 0;
281
282                 i := 1;
283                 loop
284                     value := value * 10;
285                     value := value + Char.Ord (tokenText(i)) -
286                                      Char.Ord ($0);
287
288                     exit when i = tokenLength;
289                     i := i + 1;
290                 end loop;
291
292                 if sign = negative then
293                     value :=  - value;
294                 end if;
295
296                 return (value);
297
298             end EvaluateNumber;
299
300
```

```
301         procedure ProcessDefinitions =
302
303             imports (token, GetToken, VerifyToken, GetTokenAndVerify,
304                 VerifyAndSetClass, definitionClass, nameTable, nameTop,
305                 var nameIndex, maxNames, VerifyAndEnterName, Error,
306                 EvaluateNumber, var nextValue, Signs, EnterValue,
307                 LookupName, LookupNameAndVerify);
308
309             post (token = tGoalpost or token = tEndOfFile);
310
311             { This procedure assembles the definitional part of the
312              Syntax/Semantic Language }
313
314             begin
315                 var stringSynonym:
316                     Boolean;
317
318                 { Handle Definition Classes }
319
320                 loop
321                     GetToken;
322                     exit when token = tGoalpost or
323                         token = tEndOfFile;
324
325                     { Accept Class Name }
326
327                     VerifyToken (tName);
328                     VerifyAndSetClass;
329                     GetTokenAndVerify (tColon);
330
331                     { Handle Definitions Within Class }
332
333                     GetToken;
334                     loop
335                         exit when token = tSemicolon or
336                             token = tEndOfFile;
337
338                         { Accept Defined Name }
339
340                         VerifyToken (tName);
341                         VerifyAndEnterName;
342                         GetToken;
343
344                         { Handle String Synonym if present }
345
346                         if token = tString then
347                             if definitionClass = cInput then
348                                 VerifyAndEnterName;
349                             else
350                                 Error (eIllegalString);
351                             end if;
352
```

```
353                             stringSynonym := true;
354                             GetToken;
355                         else
356                             stringSynonym := false;
357                         end if;
358
359                         { Handle Value if present }
360
361                         if token = tEqual then
362                             GetToken;
363
364                             if token = tName or token = tString then
365                                 LookupName;
366                                 nextValue(definitionClass) :=
367                                     nameTable(nameIndex).value;
368
369                             elseif token = tMinus then
370                                 GetTokenAndVerify (tNumber);
371                                 nextValue(definitionClass) :=
372                                     EvaluateNumber(negative);
373
374                             else
375                                 VerifyToken (tNumber);
376                                 nextValue(definitionClass) :=
377                                     EvaluateNumber(positive);
378                             end if;
379
380                             GetToken;
381                         end if;
382
383                         { Enter Value of Name }
384
385                         EnterValue (nameTop, nextValue(definitionClass));
386
387                         if stringSynonym then
388                             EnterValue (nameTop - 1,
389                                         nextValue(definitionClass));
390                         end if;
391
392                         nextValue(definitionClass) :=
393                                         nextValue(definitionClass) + 1;
394
395                     end loop;
396
397                 end loop;
398
399             end ProcessDefinitions;
400
401
402         procedure EmitDefinitions =
403
404             imports (var IO, var nameIndex, nameChars, nameTable,
```

```
405                 noPredefinedNames, ConvertNumber, WriteLine,
406                 nameTop, WriteString, outputFile);
407
408             { Emits Assembled Constant Definitions }
409
410             begin
411                 var i:
412                     1 .. maxNameLength;
413
414                 WriteLine;
415
416                 nameIndex := noPredefinedNames;
417                 loop
418                     exit when nameIndex = nameTop;
419                     nameIndex := nameIndex + 1;
420
421                     if nameChars (nameTable(nameIndex).charsIndex)
422                         not = quote
423                         then
424                         { A real name, not a string synonym, so
425                           output it }
426                         IO.WriteChar (outputFile, tab);
427                         WriteString ('pervasive const ');
428
429                         i := 1;
430                         loop
431                             IO.WriteChar (outputFile, nameChars
432                                                 (nameTable(nameIndex).
433                                                 charsIndex + i - 1));
434
435                             exit when i = nameTable(nameIndex).length;
436                             i := i + 1;
437                         end loop;
438
439                         WriteString (' := ');
440                         ConvertNumber (nameTable(nameIndex).value);
441                         WriteString ('; '); {rmb}
442                         WriteLine;
443                     end if;
444
445                 end loop;
446
447             end EmitDefinitions;
448
449
450         procedure Emit (value: SignedInt) =
451
452             imports (var table, var tableAddress, maxTableSize,
453                 procedureBase, maxProcedureSize, Error);
454
455             post { Value has been entered in the assembled table
456                 fragment for the current procedure };
```

```
457
458             begin
459                 if tableAddress < maxTableSize then
460                     if tableAddress - procedureBase < maxProcedureSize
461                         then
462                         table(tableAddress - procedureBase) := value;
463                         tableAddress := tableAddress + 1;
464                     else
465                         Error (eProcedureTooLarge);
466                     end if;
467
468                 else
469                     Error (eTableTooLarge);
470                 end if;
471
472             end Emit;
473
474
475
476         procedure EmitFixup (address: 0 .. maxTableSize,
477                              value: SignedInt) =
478
479             imports (var table, procedureBase);
480
481             { Performs fixups of table branch addresses }
482
483             pre (table(address - procedureBase) = nullAddress);
484
485             post (table(address - procedureBase) = value);
486
487             begin
488                 table(address - procedureBase) := value;
489
490             end EmitFixup;
491
492
493         procedure WriteProcedure =
494
495             imports (var IO, tableFile, table, tableAddress,
496                 var procedureBase, maxProcedureSize);
497
498             { Add the table fragment for the current procedure to the
499              assembled table }
500
501             begin
502                 var i:
503                     0 .. maxProcedureSize;
504
505                 i := 0;
506                 loop
507                     exit when i >= tableAddress - procedureBase;
508                     IO.WriteInt (tableFile, table(i));
```

```
509                 i := i + 1;
510             end loop;
511
512             procedureBase := tableAddress;
513
514         end WriteProcedure;
515
516
517     procedure DeleteListItem (setLabel: 0 .. maxSets,
518                               var node: SetTableEntry) =
519
520         imports (var linkTable, var setTable);
521
522         { This procedure deletes a setLabel from the list in a Set
523           TableEntry }
524
525         begin
526             var test:
527                 SignedInt;
528             var prior:
529                 SignedInt;
530
531             test := node.root;
532
533             loop
534                 exit when (linkTable (test).info = setLabel);
535
536                 prior := test;
537                 test := linkTable (test).next;
538             end loop;
539
540             if test = node.root then
541                 node.root := linkTable (test).next;
542             else
543                 linkTable (prior).next := linkTable (test).next;
544             end if;
545
546         end DeleteListItem;
547
548
549     procedure AddListItem (var node: SetTableEntry,
550                            setLabel: 0 .. maxSets) =
551
552         imports (var setTable, var linkTable, var linkTop,
553                  maxLinks);
554
555         { Adds setLabel to linked list in a SetTableEntry; list
556           is kept sorted in ascending order; duplicates are
557           eliminated; setLabel represents S/Sl rule handles }
558
559         begin
560             var test:
```

```
561                     SignedInt;  { Points to list position being
562                                   examined }
563                 var prior:
564                     SignedInt;  { Points to list position before test }
565
566                 test := node.root;
567
568                 { List is not empty; setLabel is evaluated for proper
569                   place }
570
571                 loop
572                     exit when (test = 0) or
573                               (linkTable (test).info >= setLabel);
574
575                     prior := test;
576                     test := linkTable (test).next;
577                 end loop;
578
579                 { If list is empty or setLabel is not a duplicate }
580
581                 if (test = 0) or (linkTable (test).info not =
582                     setLabel) then
583                     linkTop := linkTop + 1;
584                     linkTable (linkTop).next := test;
585                     if test = node.root then
586                         node.root := linkTop;  { Insert at list
587                                                  front }
588                     else
589                         linkTable (prior).next := linkTop;
590                               { All others }
591                     end if;
592
593                     linkTable (linkTop).info := setLabel;
594                 end if;
595
596             end AddListItem;
597
598
599         procedure AddLists (child: SetTableEntry,
600                             var parent: SetTableEntry) =
601
602             imports (AddListItem, linkTable);
603
604             { This procedure adds the list of setLabels from a child
605               node to its parent node }
606
607             begin
608                 var temp:
609                     SignedInt;
610
611                 temp := child.root;
612
```

```
613                 loop
614                     exit when (temp = 0);
615                     AddListItem (parent, linkTable (temp).info);
616                     temp := linkTable (temp).next;
617                 end loop;
618
619             end AddLists;
620
621
622         procedure AddLabelToSets (setLabel: 0 .. maxSets) =
623
624             imports (var sequenceStack, sequenceTop, AddListItem,
625                      maxSequences, sequenceLinks, setTable, nameTable,
626                      maxSets, maxLinks, var handleSet);
627
628             { This procedure adds a setLabel to each SetTableEntry
629               whose label is on the sequenceStack; it also checks if
630               setLabel should be added to handleSet }
631
632             begin
633                 var i:
634                     0 .. maxSequences;
635                 var temp:
636                     0 .. maxLinks;
637
638                 i := sequenceTop;
639
640                 loop
641                     exit when (i = 0);  { All levels have been
642                                           processed }
643
644                     temp := sequenceStack (i).root;
645
646                     { List of setLabels on a level processed }
647
648                     loop
649                         exit when (temp = 0);
650                         AddListItem (setTable(SequenceLinks
651                                                 (temp).info),
652                                      setLabel);
653                         temp := sequenceLinks (temp).next;
654                     end loop;
655
656                     i := i - 1;  { Next level of sequenceStack }
657                 end loop;
658
659                 if sequenceStack (sequenceTop).exposed then
660                     AddListItem (setTable (handleSet), setLabel);
661                     if not nameTable (nameIndex).derivedEmpty then
662                         sequenceStack (sequenceTop).exposed := false;
663                     end if;
664                 end if;
```

```
665
666             end AddLabelToSets;
667
668
669         procedure MergeBits (child: SetTableEntry,
670                              var parent: SetTableEntry) =
671
672             imports (setSize, var SetAdd, maxBits, SetIn, flagValue);
673
674             { This procedure unions a child set (which has been on a
675               list of setLabels) with its parent set }
676
677             begin
678                 var count:
679                     0 .. setSize + 1 := 0;
680                 var bitCount:
681                     0 .. maxBits;
682                 var subSet:
683                     Powerset;  { 15-bit portion of a set }
684
685                 loop
686                     exit when (count > setSize);
687
688                     subSet := child.sett (count);
689                     bitCount := 0;
690
691                     loop
692                         exit when (bitCount = maxBits);
693
694                         if SetIn (subSet, flagValue (bitCount)) then
695                             SetAdd (parent.sett (count),
696                                     flagValue (bitCount));
697                         end if;
698
699                         bitCount := bitCount + 1;
700                     end loop;
701
702                     count := count + 1;
703                 end loop;
704
705             end MergeBits;
706
707
708         procedure ReturnChildren (node: SetTableEntry,
709                                   var setLabel: 0 .. maxSets) =
710
711             imports (setTable, linkTable);
712
713             { Returns setLabel of children nodes which have visiting
714
715               flag orf or setLabel = 0 indicates all nodes have been
716               processed }
```

```
717
718             begin
719                 var found:
720                     Boolean := false;  { Indicates node with visiting
721                                          flag off }
722                 var temp:
723                     SignedInt;
724
725                 temp := node.root;
726
727                 loop
728                     exit when (temp = 0) or (found);
729
730                     if not setTable (linkTable (temp).info).visiting
731                         then
732                         setLabel := linkTable (temp).info;
733                         found := true;
734                     else
735                         temp := linkTable (temp).next;
736                     end if;
737
738                 end loop;
739
740                 if temp = 0 then  { end of list }
741                     setLabel := 0;
742                 end if;
743
744             end ReturnChildren;
745
746
747         procedure Closure (var node: SetTableEntry) =
748
749             imports (var setTable, DeleteListItem, MergeBits,
750                      AddLists);
751
752             { This procedure closes each setTableEntry of a set and
753               its list of S/SL rule handles; the list and set are
754               merged into a pure 120-bit set }
755
756             begin
757                 var child:
758                     SetTableEntry;  { Each node in list of setLabels }
759                 var setLabel:
760                     0 .. maxSets;  { Index to setTable }
761
762                 node.visiting := true;
763
764                 loop
765                     ReturnChildren (node, setLabel);
766                     exit when (setLabel = 0);  { All children visited }
767
768                     child := setTable (setLabel);
```

```
769                 DeleteListItem (setLabel,node);
770                                 { Remove setLabel }
771                 Closure (child);
772                 MergeBits (child, node);  { Add child's bits to
773                                             parent }
774                 AddLists (child, node);  { Add child's list to
775                                            parent }
776
777             end loop;
778
779             node.visiting := false;
780
781         end Closure;
782
783
784     procedure CloseSets =
785
786         imports (setTable, maxSets, setTop);
787
788         { This procedure closes the sets in setTable }
789
790         begin
791             var i:
792                 1 .. maxSets := 1;
793
794             loop
795                 Closure (setTable (i));
796                 exit when (i = setTop);
797
798                 i := i + 1;
799
800             end loop;
801
802         end CloseSets;
803
804
805     procedure TraverseList (start: SignedInt,
806                             var endOfList: SignedInt) =
807
808         imports (sequenceLinks);
809
810         { This procedure locates the end of a list }
811
812         begin
813             var temp:
814                 SignedInt ;
815             var prior:
816                 SignedInt;
817
818             temp := start;
819             loop
820                 exit when (temp = 0);
```

```
821
822                     prior := temp;
823                     temp := sequenceLinks (temp).next;
824
825                 end loop;
826
827                 endOfList := prior;
828
829             end TraverseList;
830
831
832         procedure AddToSequence (setLabel: 0 .. maxSets) =
833
834             imports (var sequenceStack, sequenceTop, var seqLinkTop,
835                      maxLinks, var sequenceLinks, TraverseList);
836
837             { This procedure adds a setLabel to a list of setLabels on
838               the sequenceStack }
839
840             begin
841                 var endOfList:
842                     SignedInt;
843                 var temp:
844                     SignedInt;
845
846                 temp := sequenceStack (sequenceTop).root;
847                 seqLinkTop := seqLinkTop + 1;
848
849                 if temp = 0 then  { Empty sequence level }
850                     sequenceStack (sequenceTop).root := seqLinkTop;
851                 else
852                     TraverseList (temp, endOfList);
853                     sequenceLinks (endOfList).next := seqLinkTop;
854                 end if;
855
856                 sequenceLinks (seqLinkTop).next := 0;
857                 sequenceLinks (seqLinkTop).info := setLabel;
858
859             end AddToSequence;
860
861
862         procedure MoveChoiceSequence (var list: SignedInt,
863                                       listAddition: SignedInt) =
864
865             imports (var choiceStack, var sequenceStack, TraverseList);
866
867             { This procedure moves a choiceSequence: 1) to the
868               choiceTable temporarily or 2) back to the
869               sequenceStack }
870
871             begin
872                 var endOfList:
```

```
873                     SignedInt;  { End of a list of setLabels }
874
875                 if list = 0 then  { no list started }
876                     list := listAddition;
877                 else
878                     TraverseList (list, endOfList);
879                     sequenceLinks (endOfList).next := listAddition;
880                 end if;
881
882             end MoveChoiceSequence;
883
884
885         procedure AddBit (var sett: SetType, bitNo: SignedInt) =
886
887             imports (var setTable, setSize, Flag, SetType, SetAdd,
888                      maxBits, flagValue);
889
890             { Adds a single bit to a set in SetTableEntry }
891
892             begin
893                 var wordNo:
894                     0 .. setSize;
895                 var bit:
896                     0 .. maxBits - 1;
897
898
899                 wordNo := bitNo div maxBits;
900                 bit := bitNo mod maxBits;
901                 SetAdd (sett (wordNo), flagValue (bit));
902
903             end AddBit;
904
905
906         procedure AddTokenToSets (token: SignedInt) =
907
908             imports (var sequenceStack, sequenceTop, AddBit, maxLinks,
909                      maxSequences, sequenceLinks, setTable,
910                      var handleSet);
911
912             { This procedure adds a token to each set whose label is
913               on the sequenceStack; it also checks if token should be
914               added to handleSet }
915
916             begin
917                 var i:
918                     0 .. maxSequences;
919                 var temp:
920                     0 .. maxLinks;
921
922                 i := sequenceTop;
923
924                 loop
```

```
925                     exit when (i = 0);  { All levels have been
926                                           processed }
927
928                     temp := sequenceStack (i).root;
929                     loop
930                         exit when (temp = 0);
931                         AddBit (setTable (sequenceLinks
932                                           (temp).info).sett, token);
933                         temp := sequenceLinks (temp).next;
934                     end loop;
935
936                     i := i - 1;
937
938                 end loop;
939
940                 if sequenceStack (sequenceTop).exposed then
941                     AddBit (setTable (handleSet).sett,token);
942                     sequenceStack (sequenceTop).exposed := false;
943                 end if;
944
945             end AddTokenToSets;
946
947
948         procedure ClearSet (setLabel: 0 .. maxSets) =
949
950             imports (var setTable, setSize, SetType, SetClear);
951
952             { This procedure empties a SetTableEntry: sets bits to 0
953               and root to 0 }
954
955             begin
956                 var i:
957                     SignedInt := 0;
958                 var temp:
959                     SetType;
960
961                 setTable (setLabel).root := 0;  { empty list }
962                 setTable (setLabel).visiting := false;  { closure }
963                 temp := setTable (setLabel).sett;
964
965                 loop
966                     SetClear (temp (i));
967                     exit when (i = setSize);
968
969                     i := i + 1;
970
971                 end loop;
972
973             end ClearSet;
974
975
976         procedure EnterLabel (nameIndex: 0 .. maxNames,
```

```
977                                 labelValue: 0 .. maxSets) =
978
979             imports (var nameTable);
980
981             { This procedure enters labelValue as location in setTable
982               for the handleSet of the S/SL procedure in nameTable
983               (nameIndex) }
984
985             begin
986
987             if labelValue < 1 then BREAKPNT (1500); end if;
988                 nameTable (nameIndex).setLabel := labelValue;
989                 nameTable (nameIndex).derivedEmpty := true;
990
991             end EnterLabel;
992
993
994         procedure GetSet (var setLabel: 0 .. maxSets) =
995
996             imports (var setTop, maxSets, ClearSet, AddToSequence);
997
998             { This procedure assigns a new empty set to an operator
999               token }
000
001             begin
002
003                 if setTop < maxSets then
004                     setTop := setTop + 1;
005                     setLabel := setTop;
006                     ClearSet (setLabel);
007                 else
008                     Error (eTooManySets);
009                 end if;
010
011             end GetSet;
012
013
014         procedure PushSequenceStack =
015
016             imports (var sequenceTop, var sequenceStack, maxSequences,
017                      Error);
018
019             { For error recovery: procedure starts a  new sequence }
020
021
022             begin
023                 if sequenceTop < maxSequences then
024                     sequenceTop := sequenceTop + 1;
025                     sequenceStack (sequenceTop).root := 0;
026                     sequenceStack (sequenceTop).exit := false;
027                     if choiceTop > 0 then
028                         sequenceStack (sequenceTop).exposed :=
```

```
029                         choiceStack (choiceTop).initExposed;
030                 else
031                     sequenceStack (sequenceTop).exposed := true;
032                 end if;
033             else
034                 Error (eSequencesTooDeep);
035             end if;
036
037         end PushSequenceStack;
038
039
040     procedure PopSequenceStack =
041
042         imports (var sequenceTop, var sequenceStack, maxSequences,
043                  MoveChoiceSequence, choiceStack, choiceTop,
044                  sequenceLinks, AddListItem, setTable);
045
046         { Error recovery: this procedure removes a sequence level }
047
048         begin
049             var temp:
050                 SignedInt;
051
052             if  ((cycletop > 0) and
053                 (not sequenceStack (sequenceTop).exit)) then
054
055                 { Choice alternative inside a cycle did not
056                  terminate with return symbol; add cycleSet to
057                  each set on top level of sequenceStack }
058
059                 temp := sequenceStack (sequenceTop).root;
060
061                 loop
062                     exit when (temp = 0);
063                     AddListItem (setTable (sequenceLinks
064                                            (temp).info),
065                                  cycleStack (cycleTop).cycleSet);
066                     temp := sequenceLinks (temp).next;
067                 end loop;
068             end if;
069
070             if choiceTop > 0 then  { choice alternatives being
071                                      processed }
072
073                 { List of setLabels for choice alternative are
074                   temporarily moved to choiceStack }
075
076                 MoveChoiceSequence (choiceStack (choiceTop).root,
077                                     sequenceStack (sequenceTop).root);
078                 if sequenceStack (sequenceTop).exposed then
079                     choiceStack (choiceTop).finalExposed := true;
080                 end if;
```

```
081                 end if;
082
083                 sequenceTop := sequenceTop - 1;
084
085             end PopSequenceStack;
086
087
088
089         procedure PushCycle  (setLabel: 0 .. maxSets) =
090
091             imports (var cycleTop, var cycleStack, maxCycles,
092                 exitTop, tableAddress, Error, token);
093
094             pre (token = tCycle);
095
096             post { (cycleTop > 0 and cycleStack(cycleTop).exitIndex =
097                                             exitTop and
098                 cycleStack(cycleTop).cycleAddress = tableAddress) or
099                 error message has been emitted };
100
101             { This procedure processes the beginning of a cycle }
102
103             begin
104                 if cycleTop < maxCycles then
105                     cycleTop := cycleTop + 1;
106                     cycleStack(cycleTop).exitIndex := exitTop;
107                     cycleStack(cycleTop).cycleAddress := tableAddress;
108                     cycleStack(cycleTop).cycleSet := setLabel;
109                     cycleStack (cycleTop).initExposed := sequenceStack
110                                                 (sequenceTop).exposed;
111                     cycleStack (cycleTop).finalExposed := false;
112                 else
113                     Error (eCyclesTooDeep);
114                 end if;
115
116             end PushCycle;
117
118
119
120         procedure EnterCycleExit  =
121
122             imports (var exitTop, var exitAddress, maxExits,
123                 tableAddress, cycleTop, Error, token);
124
125             pre (token = tExit and cycleTop > 0);
126
127             post { (exitTop > 0 and exitAddress(exitTop) =
128                     tableAddress - 1) or
129                 Error message has been Emitted };
130
131             { This procedure processes cycle exits }
132
```

```
133             begin
134                 if exitTop < maxExits then
135                     exitTop := exitTop + 1;
136                     exitAddress(exitTop) := tableAddress - 1;
137                 else
138                     Error (eTooManyExits);
139                 end if;
140
141             end EnterCycleExit;
142
143
144         procedure PopCycle  =
145
146             imports (var cycleTop, cycleStack, var exitTop,
147                 exitAddress, tableAddress, EmitFixup, token);
148
149             pre (token = tCycleEnd and cycleTop > 0);
150
151             post { Cycle exits have  fixed up and cycle stack has been
152                    been popped };
153
154             begin
155                 { Fixup cycle repeat }
156                 EmitFixup (tableAddress - 1, cycleStack(cycleTop).
157                                                     cycleAddress);
158
159                 {Fixup cycle exits}
160                 loop
161                     exit when exitTop  =  cycleStack(cycleTop).
162                                                         exitIndex;
163                     EmitFixup (exitAddress(exitTop), tableAddress);
164                     exitTop := exitTop - 1;
165                 end loop;
166
167                 { HandleSets: containing sequence gets same exposed
168                   value }
169
170                 sequenceStack (sequenceTop).exposed :=
171                                 cycleStack (cycleTop).finalExposed;
172
173                 { Pop cycle stack }
174                 cycleTop := cycleTop - 1;
175
176             end PopCycle;
177
178
179         procedure PushChoice (pushClass: NameClasses) =
180
181             imports (var choiceTop, var choiceStack, maxChoices,
182                 tableAddress, mergeTop, labelTop, Error, token);
183
184             pre (token = tInput or token = tName);
```

```
185
186             post { (choiceTop  >  0 and
187                 choiceStack(choiceTop).choiceClass = pushClass and
188                 choiceStack(choiceTop).mergeIndex = mergeTop and
189                 choiceStack(choiceTop).labelIndex = labelTop and
190                 choiceStack(choiceTop).choiceAddress = tableAddress
191                                                             - 1 and
192                 choiceStack(choiceTop).otherwisePresent = false and
193                 choiceStack(choiceTop).firstAlternative = true) or
194                 Error message has been Emitted };
195
196             { This procedure processes the beginning of a choice }
197
198             begin
199                 if choiceTop < maxChoices then
200                     choiceTop := choiceTop + 1;
201                     choiceStack(choiceTop).choiceClass := pushClass;
202                     choiceStack(choiceTop).choiceAddress :=
203                         tableAddress - 1;
204                     choiceStack(choiceTop).mergeIndex := mergeTop;
205                     choiceStack(choiceTop).labelIndex := labelTop;
206                     choiceStack(choiceTop).otherwisePresent := false;
207                     choiceStack(choiceTop).firstAlternative := true;
208                     choiceStack(choiceTop).root := 0;
209                     choiceStack (choiceTop).initExposed :=
210                         sequenceStack(sequenceTop).exposed;
211                     choiceStack (choiceTop).finalExposed := false;
212                 else
213                     Error(eChoicesTooDeep);
214                 end if;
215
216             end PushChoice;
217
218
219         procedure EnterChoiceMerge =
220
221             imports (var mergeTop, var mergeAddress, maxMerges,
222                 tableAddress, choiceTop, Error, token);
223
224             pre ((token = tOr or token = tChoiceEnd) and
225                   choiceTop > 0);
226
227             post { (mergeTop > 0 and mergeAddress(mergeTop) =
228                 tableAddress - 1) or Error message has been Emitted };
229
230             begin
231                 if mergeTop < maxMerges then
232                     mergeTop := mergeTop + 1;
233                     mergeAddress(mergeTop) := tableAddress - 1;
234                 else
235                     Error(eTooManyMerges);
236                 end if;
```

```
237
238             end EnterChoiceMerge;
239
240
241
242         procedure EnterChoiceLabel (value: SignedInt) =
243
244             imports (var labelTop, var labelAddress, var labelValue,
245                 maxLabels, tableAddress, Error, choiceTop);
246
247             pre (choiceTop > 0);
248
249             post { (labelTop > 0 and labelValue(labelTop) =  value and
250                 labelAddress(labelTop) = tableAddress) or
251                 Error message has been Emitted };
252
253             begin
254                 if labelTop < maxLabels then
255                     labelTop := labelTop + 1;
256                     labelValue(labelTop) := value;
257                     labelAddress(labelTop) := tableAddress;
258                 else
259                     Error(eTooManyLabels);
260                 end if;
261
262             end EnterChoiceLabel;
263
264
265         procedure ProcessChoiceOtherwise =
266
267             imports (choiceTop, var choiceStack, var labelTop,
268                 labelValue, maxLabels, Emit, EmitFixup, tableAddress,
269                 token, labelAddress);
270
271             pre (token = tOtherwise or token = tChoiceEnd);
272
273             post { Choice table address has been fixed up and
274                 choice table has been Emitted. };
275
276             begin
277                 var i:
278                     1 .. maxLabels;
279
280                 { Fixup choice table address }
281
282                 EmitFixup (choiceStack(choiceTop).choiceAddress,
283                           tableAddress);
284
285                 { Emit choice table }
286
287                 Emit (labelTop - choiceStack(choiceTop).labelIndex);
288                       { Number of entries }
```

```
289
290                 i := choiceStack(choiceTop).labelIndex;
291                 loop
292                     exit when i = labelTop;
293                     i := i + 1;
294                     Emit (labelValue(i));
295                     Emit (labelAddress(i));
296                 end loop;
297
298                 labelTop := choiceStack(choiceTop).labelIndex;
299                 choiceStack(choiceTop).otherwisePresent := true;
300
301             end ProcessChoiceOtherwise;
302
303
304         procedure PopChoice =
305
306             imports (var choiceTop, choiceStack, var mergeTop, mergeAddress
307                 EmitFixup, tableAddress, token);
308
309             pre (token = tChoiceEnd);
310
311             post { Choice merges have been fixed up and choice stack popped
312
313             begin
314                 { Fix choice merges }
315                 loop
316                     exit when mergeTop = choiceStack(choiceTop).
317                                                 mergeIndex;
318                     EmitFixup (mergeAddress(mergeTop), tableAddress);
319                     mergeTop := mergeTop - 1;
320                 end loop;
321
322                 sequenceStack (sequenceTop).exposed := choiceStack
323                                             (choiceTop).finalExposed;
324                 { Pop choice stack }
325                 choiceTop := choiceTop - 1;
326
327             end PopChoice;
328
329
330         procedure ProcessProcedure =
331
332             imports (token, GetToken, VerifyToken, GetTokenAndVerify,
333                 Signs, LookupName, VerifyNameClass,
334                 EvaluateNumber, var nameTable, nameTop, nameIndex,
335                 var callTable, PushCycle, EnterCycleExit, PopCycle,
336                 PushChoice, ProcessChoiceOtherwise, PopChoice,
337                 var choiceStack, var choiceTop, labelValue,
338                 Emit, tableAddress, Error, var setTable, maxCalls,
339                 GetSet, AddTokenToSets, AddLabelToSets, labelAddress,
340                 var sequenceStack, AddToSequence, EnterLabel,
```

```
341                 PopSequenceStack, maxSets, lineNo, NameClasses,
342                 VerifyAndEnterName, var callTop, var cycleTop,
343                 WriteProcedure, EnterChoiceLabel, PushSequenceStack,
344                 NameClasses, LookupNameAndVerify, var callTop,
345                 EnterChoiceMerge, WriteProcedure, maxCalls,
346                 MovechoiceSequence, maxSets, lineNo);
347
348             pre (token = tColon);
349
350             post (token = tSemicolon or token = tEndOfFile);
351
352             { This procedure assembles a Syntax/Semantic Language
353               procedure }
354
355             begin
356                 var setLabel:
357                     0 .. maxSets;  { Index to setTable }
358                 var saveName:
359                     0 .. maxNames;  { Saves index for procedure name }
360
361                 saveName := nameIndex;
362                 PushSequenceStack;
363
364                 loop
365                     GetToken;
366                     exit when token = tSemicolon or
367                               token = tEndOfFile;
368
369                     case token of
370
371                         tInput =>
372                             { Process input operation }
373                             Emit (oInput);
374                             GetTokenAndVerify (tName);
375                             LookupNameAndVerify (cInput);
376                             Emit (nameTable(nameIndex).value);
377                             Emit (lineNo);  { S/SL line number }
378                             AddTokenToSets (nameTable (nameIndex).
379                                                             value);
380                             GetSet (setLabel);
381                             Emit (setLabel);
382                             AddToSequence (setLabel);
383                             end tInput;
384
385                         tString =>
386                             { Process input operation by string }
387                             Emit (oInput);
388                             LookupNameAndVerify (cInput);
389                             Emit (nameTable(nameIndex).value);
390                             Emit (lineNo);
391                             AddTokenToSets (nameTable (nameIndex).
392                                                         value);
```

```
393                         GetSet (setLabel);
394                         Emit (setLabel);
395                         AddToSequence (setLabel);
396                         end tString;
397
398                     tOutput =>
399                         { Process output operation }
400                         Emit (oEmit);
401                         GetTokenAndVerify (tName);
402                         LookupName;
403
404                         if nameTable(nameIndex).class not =
405                                                     cOutput then
406                             { Allow inputs to be output also }
407                             VerifyNameClass (cInput);
408                         end if;
409
410                         Emit (nameTable(nameIndex).value);
411                         end tOutput;
412
413                     tError =>
414                         { Process error operation }
415                         Emit (oError);
416                         GetTokenAndVerify (tName);
417                         LookupNameAndVerify (cError);
418                         Emit (nameTable(nameIndex).value);
419                         end tError;
420
421                     tCall =>
422                         { Process procedure call }
423                         Emit (oCall);
424                         GetTokenAndVerify (tName);
425                         LookupName;
426
427                         if nameIndex = notFound then
428                             VerifyAndEnterName;
429                             GetSet (setLabel);
430                             EnterLabel (nameIndex, setLabel);
431                         else
432                             VerifyNameClass (cProcedure);
433                         end if;
434
435                         AddLabelToSets (nameTable (nameIndex).
436                                                     setLabel);
437                         Emit (nullAddress);
438
439                         if callTop < maxCalls then
440                             callTop := callTop + 1;
441                             callTable (callTop).callAddress :=
442                                 tableAddress - 1;
443                             callTable (callTop).procedureIndex :=
444                                                     nameIndex;
```

```
445                         else
446                             error (eTooManyCalls);
447                         end if;
448
449                         GetSet (setLabel);
450                         Emit (setLabel);
451                         AddToSequence (setLabel);
452
453
454                         end tCall;
455
456                     tReturn =>
457                         { Process procedure return }
458                         Emit (oReturn);
459                         end tReturn;
460
461                     tName =>
462                         { Process simple op, emitting op or
463                           parameterized op }
464                         LookupName;
465
466                         if nameTable(nameIndex).class = cSimpleOp
467                                 or nameTable(nameIndex).class =
468                                 cEmitOp then
469                             Emit (nameTable(nameIndex).value);
470
471                         elseif nameTable(nameIndex).class =
472                             cParmOp then
473                             Emit (nameTable(nameIndex).value);
474                             GetTokenAndVerify (tLeftParen);
475                             GetToken;
476
477                             if token = tName then
478                                 LookupNameAndVerify (cValue);
479                                 Emit (nameTable(nameIndex).value);
480
481                             elseif token = tMinus then
482                                 GetTokenAndVerify (tNumber);
483                                 Emit (EvaluateNumber (negative));
484
485                             else
486                                 VerifyToken (tNumber);
487                                 Emit (EvaluateNumber (positive));
488                             end if;
489
490                             GetTokenAndVerify (tRightParen);
491
492                         else
493                             { Force error }
494                             VerifyNameClass (cSimpleOp);
495                         end if;
496
```

```
497                         end tName;
498
499                     tCycle =>
500                         { Process cycle head }
501                         GetSet (setLabel);
502                         AddToSequence (setLabel);
503                         PushCycle (setLabel);
504                         end tCycle;
505
506                     tExit =>
507                         { Process cycle exit }
508                         if cycleTop > 0 then
509                             Emit (oMerge);
510                             Emit (nullAddress);
511                             sequenceStack (sequenceTop).exit :=
512                                                             true;
513                             if sequenceStack (sequenceTop).exposed
514                                 then
515                                 cycleStack (cycleTop).finalExposed
516                                                         := true;
517                             end if;
518                             EnterCycleExit;
519                         else
520                             Error (eSyntaxError);
521                         end if;
522
523                         end tExit;
524
525                     tCycleEnd =>
526                         { Process cycle end }
527                         if cycleTop > 0 then
528                             Emit (oRepeat);
529                             Emit (nullAddress);
530                             PopCycle;
531                         else
532                             Error (eSyntaxError);
533                         end if;
534
535                         end tCycleEnd;
536
537                     tChoice =>
538                         { Process choice head }
539                         GetToken;
540
541                         if token = tInput then
542                             { Input choice }
543                             Emit (oInputChoice);
544                             Emit (nullAddress);
545                             PushChoice (cInput);
546                             Emit (lineNo);
547                             GetSet (setLabel);
548                             Emit (setLabel);
```

```
549                         AddToSequence (setLabel);
550                     else
551                         { Semantic Choice }
552                         VerifyToken (tName);
553                         LookupNameAndVerify (cChoiceOp);
554                         Emit (nameTable(nameIndex).value);
555                         Emit (nullAddress);
556                         PushChoice (cValue);
557                     end if;
558
559                     end tChoice;
560
561                 tOr =>
562                     { Process next alternative }
563                     if choiceTop > 0 then
564                         if not choiceStack(choiceTop).
565                             firstAlternative then
566                             { Emit Choice merge  }
567                             Emit (oMerge);
568                             Emit (nullAddress);
569                             EnterChoiceMerge;
570                             PopSequenceStack;
571                         else
572                             choiceStack(choiceTop).
573                                 firstAlternative := false;
574                         end if;
575
576                         PushSequenceStack;
577
578                         { Process alternative label list }
579                         loop
580                             GetToken;
581
582                             if token = tName then
583                                 LookupNameAndVerify
584                                     (choiceStack
585                                      (choiceTop).choiceClass);
586                                 EnterChoiceLabel
587                                     (nameTable(nameIndex).
588                                                 value);
589                                 AddTokenToSets (nameTable
590                                           (nameIndex).value);
591                                 sequenceStack (sequenceTop).
592                                           exposed := false;
593
594                             elseif token = tString then
595                                 LookupNameAndVerify
596                                     (choiceStack
597                                      (choiceTop).choiceClass);
598                                 EnterChoiceLabel
599                                     (nameTable(nameIndex).
600                                                 value);
```

```
601                             AddTokenToSets (nameTable
602                                             (nameIndex).
603                                                 value);
604                             sequenceStack (sequenceTop).
605                                         exposed := false;
606
607                         elseif token = tMinus then
608                             GetTokenAndVerify (tNumber);
609                             EnterChoiceLabel
610                                 (EvaluateNumber(negative));
611
612                         elseif token = tNumber then
613                             EnterChoiceLabel
614                                 (EvaluateNumber(positive));
615
616                         elseif token = tOtherwise then
617                             ProcessChoiceOtherwise;
618                             GetTokenAndVerify (tColon);
619                             exit;
620
621                         else
622                             Error (eSyntaxError);
623                         end if;
624
625                         GetToken;
626
627                         exit when token =  tColon or
628                                   token = tEndOfFile;
629                         VerifyToken (tComma);
630                     end loop;
631
632                 else
633                     Error (eSyntaxError);
634                 end if;
635
636                 end tOr;
637
638             tChoiceEnd =>
639                 { Process choice end }
640                 if choiceTop > 0 then
641
642                     if not choiceStack(choiceTop).
643                         otherwisePresent then
644                         Emit (oMerge);
645                         Emit (nullAddress);
646                         EnterChoiceMerge;
647                         ProcessChoiceOtherwise;
648
649                         if choiceStack(choiceTop).
650                             choiceClass = cInput then
651                             { Emit code to enable syntax
652                               error recovery }
```

```
653                                     Emit (oRepeat);
654                                     Emit (labelAddress
655                                           choiceStack(choiceTop).
656                                           labelIndex + 1));
657                                 else
658                                     { Emit code to catch semantic
659                                       choice failures }
660                                     Emit (oChoiceEnd);
661                                 end if;
662                             end if;
663                             PopSequenceStack;
664
665                             { List of setLabels from choiceStack
666                               is moved back to sequenceStack after
667                               processing all alternative choices }
668
669                             MoveChoiceSequence
670                                 (sequenceStack(sequenceTop).root,
671                                  choiceStack(choicetop).root);
672
673                             PopChoice;
674                         else
675                             Error (eSyntaxError);
676                         end if;
677
678                         end tChoiceEnd;
679
680                     otherwise =>
681                         Error (eSyntaxError);
682                 end case;
683
684             end loop;
685
686             { Emit procedure return }
687
688             Emit (oReturn);
689
690             { Write procedure to assembled table }
691
692             WriteProcedure;
693
694             { Verify cycle and case stacks }
695
696             if cycleTop not = 0 then
697                 Error (eMissingCycleEnd);
698                 cycleTop := 0;
699             end if;
700
701             if choiceTop not = 0 then
702                 Error (eMissingChoiceEnd);
703                 choiceTop := 0;
704             end if;
```

```
705
706                 if not sequenceStack (sequenceTop).exposed then
707                     nameTable (saveName).derivedEmpty := false;
708                 end if;
709
710                 PopSequenceStack;
711
712             end ProcessProcedure;
713
714
715         procedure ProcessProcedures =
716
717             imports (var IO, tableFile, token, var definitionClass,
718                 GetToken, VerifyToken, VerifyAndEnterName, EnterValue,
719                 GetTokenAndVerify, ProcessProcedure, Error, nameTable,
720                 handleSet, maxSets, var saveTableAddress, LookupName,
721                 VerifyNameClass, nameIndex, tableAddress);
722
723             pre (token = tGoalpost or
724                 token = tEndOfFile);
725
726             post (token = tEndOfFile);
727
728             begin
729                 var setLabel:
730                     0 .. maxSets;
731
732                 IO.Open (tableFile, outFile);
733                 definitionClass := cProcedure;
734
735                 loop
736                     exit when token = tEndOfFile;
737
738                     GetToken;
739                     exit when token = tGoalpost;
740
741                     VerifyToken (tName);
742                     LookupName;
743
744                     if nameIndex = notFound then
745                         VerifyAndEnterName;
746                         GetSet (setLabel);
747                         EnterLabel (nameIndex, setLabel);
748                     else
749                         VerifyNameClass (cProcedure);
750
751                         if nameTable(nameIndex).value not = nullValue
752                             then
753                             Error (eDoubleDefinition);
754                         end if;
755                     end if;
756
```

```
757
758                     handleSet := nameTable (nameIndex).setLabel;
759                     EnterValue (nameIndex, tableAddress);
760                     GetTokenAndVerify (tColon);
761                     ProcessProcedure;
762                 end loop;
763
764                 if token = tEndOfFile then
765                     Error (ePrematureEndOfFile);
766                 else
767                     GetTokenAndVerify (tEndOfFile);
768                 end if;
769
770                 IO.Close (tableFile);
771                 saveTableAddress := tableAddress;
772
773             end ProcessProcedures;
774
775
776         procedure EmitCode (index: 0 .. maxNames) =
777
778             imports (var IO, nameTable, nameChars, maxNameChars,
779                      outputFile, stringLength);
780
781             { Prints assembler code/rule string in nameChars }
782
783             begin
784                 var count:
785                     1 .. stringLength + 1 := 1;
786                 var string:
787                     array 1 .. stringLength of Char;
788                     { String from nameChars (only first 15 chars) }
789                 var start:
790                     0 .. maxNameChars + 1;
791                 var j:
792                     0 .. maxNames;
793
794                 j := index;
795                 if j = 0 then  { Value not found in nameTable }
796                     string := 'Error:not found';
797                 else  { Valid index to nameTable }
798                     string := '               ';
799                     start := nameTable (j).charsIndex;
800
801                     loop
802                         exit when (count > nameTable (j).length) or
803                                   (count > stringLength);
804                         string (count) := nameChars (start);
805                         start := start + 1; { next letter }
806                         count := count + 1; { next letter }
807                     end loop;
808                 end if;
```

```
809
810                 { Emit characters }
811
812                 count := 1;
813                 loop
814                     exit when (count > stringLength);
815                     IO.WriteChar (outputFile, string (count));
816                     count := count + 1;
817                 end loop;
818
819             end  EmitCode;
820
821         procedure LocateString (i: SignedInt,
822                                 opClass: NameClasses,
823                                 var index: 0 .. maxNames) =
824
825             imports (nameTable, maxNames);
826
827
828             { Linear search through nameTable using value and class
829               to locate entry point in nameChars }
830
831             begin
832                 var j:
833                     0 .. maxNames + 1 := 0;
834                 var found:
835                     Boolean := false;
836
837                 loop
838                     exit when (found) or (j > maxNames);
839                     if (nameTable (j).class  = opClass) and
840                        (nameTable (j).value = i) then
841                         found := true;
842                         index := j;  { Place in nameTable }
843                     else
844                         j := j + 1;
845                     end if;
846                 end loop;
847
848                 if not found then
849                     index := notFound;  { 0 }
850                 end if;
851
852             end LocateString;
853
854         procedure CheckForRule (i: 0 .. maxTableSize) =
855
856             imports (NameClasses, blanks, notFound,
857                      EmitCode, WriteString, LocateString, maxNames);
858
859             { Checks for beginning of S/SL rule }
860
```

```
861             begin
862                 var index:
863                     0 .. maxNames;
864
865                 LocateString (i, cProcedure, index);
866                 if index = notFound then
867                     WriteString (blanks);  { Pad with blanks }
868                 else
869                     EmitCode (index);  { Print rule }
870                     WriteString ('      ');
871                 end if;
872
873
874             end CheckForRule;
875
876         procedure Resolve (i: 0 .. maxTableSize,
877                            var j: 1 .. maxCalls + 1,
878                            var operand: SignedInt) =
879
880             imports (callTop, callTable, nullAddress, nameTable);
881
882             { Resolves null addresses in nameTable }
883
884             begin
885                 if (j <= callTop) and
886                    (i = callTable (j).callAddress) then
887                     assert (operand = nullAddress);
888                     operand := nameTable (callTable (j).
889                                procedureIndex).value;
890                     j := j + 1;
891                 end if;
892
893             end Resolve;
894
895         procedure FormatTable (var i: 0 .. maxTableSize,
896                                var j: 0 .. maxCalls + 1,
897                                var operand: SignedInt) =
898
899
900             imports (blanks, WriteString, tableFile, var IO,
901                      Resolve, ConvertNumber);
902
903             { Prints actual table integers }
904
905             begin
906
907                 WriteString (blanks);
908                 IO.ReadInt (tableFile, operand);
909                 i := i + 1;  { Next place in table }
910                 Resolve (i, j, operand); { check for unresolved
911                                                        address }
912                 ConvertNumber (operand); { print operand }
```

```
913
914             end FormatTable;
915
916
917
918         procedure TranslateSetLabel (var i: 0 .. maxTableSize) =
919
920             imports (WriteString, blanks, setTable, ConvertNumber,
921                      var IO, tableFile);
922
923             { This procedure reads a setLabel from the tableFile
924               and translates it to an actual address in the table }
925
926             begin
927                 var setLabel:
928                     SignedInt;
929
930                 WriteString (blanks);
931                 IO.ReadInt (tableFile, setLabel);
932                 i := i + 1;  { Next place in table }
933                 { Print tableAddress where set begins }
934                 ConvertNumber (setTable (setLabel).root);
935
936            end TranslateSetLabel;
937
938
939         procedure PrintOperands (var i: 0 .. maxTableSize,
940                                  var j: 0 .. maxCalls + 1) =
941
942             imports (WriteString, blanks, FormatTable,
943                      TranslateSetLabel);
944
945             { This procedure prints the last two opers for input and
946               and inputchoice operators: S/SL input line number and
947               setLabel }
948
949             begin
950                 var operand:
951                     SignedInt;
952
953                 WriteString (' }');
954                 WriteLine;
955                 FormatTable (i, j, operand);  { Print S/SL line number }
956                 WriteString (',  ');
957                 WriteLine;
958                 TranslateSetLabel (i);  { Print actual set address }
959                 WriteString (',  {');
960                 WriteString (blanks);
961                 WriteString ('                ');
962
963             end PrintOperands;
964
```

```
965
966
967
968         procedure ProcessOperands (opCode: SignedInt,
969                                    var i: 0 .. maxTableSize,
970                                    var j: 1 .. maxCalls + 1) =
971
972             imports (WriteLine, NameClasses, maxNames,
973                      LocateString, , EmitCode, var IO,
974                      var inputChoiceTableTop, choiceTable, blanks,
975                      ConvertNumber, WriteString, TranslateSetLabel
976                      FormatTable, PrintOperands);
977
978             { Processes operands for table printing }
979
980             begin
981                 var operand:
982                     SignedInt;
983                 var index:
984                     0 .. maxNames;
985
986                 FormatTable (i, j, operand); { Print operand }
987                 WriteString (',  {                        ');
988
989                 case opCode of
990
991                     oCall =>
992                         LocateString (operand, cProcedure, index);
993                         EmitCode (index);
994                         WriteString (' }');
995                         WriteLine;
996                         TranslateSetLabel (i);  { Print actual set
997                                                   address }
998                         WriteString (',  {');
999                         WriteString (blanks);
000                         WriteString ('                ');
001                         end oCall;
002
003                     oInput =>
004                         LocateString (operand, cInput, index);
005                         EmitCode (index);
006                         PrintOperands (i, j);
007                         end oInput;
008
009                     oEmit =>
010                         LocateString (operand, cOutput, index);
011                         EmitCode (index);
012                         end oEmit;
013
014                     oError =>
015                         LocateString (operand, cError, index);
016                         EmitCode (index);
```

```
017                         end oError;
018
019                     oRepeat, oMerge =>
020                         WriteString ('               ');
021                         { no string name }
022                         end oRepeat;
023
024                     oInputChoice =>
025                         inputChoiceTableTop :=
026                             inputChoiceTableTop + 1;
027                         choiceTable (inputChoiceTableTop) :=
028                             operand; { Table address saved }
029                         WriteString ('TABLE          ');
030                         PrintOperands (i, j);
031                         end oInputChoice;
032
033                 end case;
034
035                 WriteString (' }');
036                 WriteLine;
037
038             end ProcessOperands;
039
040         procedure EmitOpCode (opString: AssemCode) =
041
042             imports (var IO, codeLength, outputFile);
043
044             { Prints opCode string from opCode table }
045
046             begin
047                 var count: SignedInt;
048
049                 count := 1;
050
051                 loop
052                     exit when count > codeLength;
053                     IO.WriteChar (outputFile, opString (count));
054                     count := count + 1; { next letter }
055                 end loop;
056
057                 WriteString ('    ');
058
059             end EmitOpCode;
060
061         procedure ProcessInputChoiceTable
062             (var i: 0 .. maxTableSize, var j: 1 .. maxCalls + 1,
063              numChoices: SignedInt) =
064
065             imports (var IO, maxNames, ConvertNumber, Resolve,
066                      LocateString, EmitCode, WriteLine,
067                      nameClasses,  WriteString, FormatTable);
068
```

```
069             { Prints string f number of choices and processes pairs of
070               of choices }
071
072             begin
073                 var address:
074                     SignedInt;
075                 var count:
076                     SignedInt;
077                 var index:
078                     0 .. maxNames;
079                 var inToken:
080                     SignedInt;
081
082                 WriteString ('TABLE BEGINS    }');
083                 WriteLine; { end of line }
084                 count := 1;
085
086                 loop  { through pairs of choices }
087                     exit when count > numChoices;
088                     FormatTable (i, j, inToken);  { Print inToken }
089                     WriteString (',  {                          ');
090                     LocateString (inToken, cInput, index);
091                     EmitCode (index);
092                     WriteString (' }');
093                     WriteLine;
094                     FormatTable (i, j, address); { print address
095                                                              label }
096                     WriteString (', ');
097                     WriteLine;
098                     count := count + 1; { next pair }
099                 end loop;
100
101             end ProcessInputChoiceTable;
102
103
104         procedure FindSemanticString (opCode: SignedInt,
105                                       var index: 0 .. maxNames,
106                                       var opClass: NameClasses) =
107
108             imports (nameTable, NameClasses, maxNames);
109
110             { Search nameTable for value = opCode and class =
111               cSimpleOp,cParmOp, cChoiceOp, or cEmitOp }
112
113             begin
114                 var j:
115                     0 .. maxNames + 1 := 0;
116                 var found:
117                     Boolean := false;
118
119                 loop
120                     exit when (found) or (j > maxNames);
```

```
121                     if ((nameTable (j).class = cEmitOp) or
122                         (nameTable (j).class = cSimpleOp) or
123                         (nameTable (j).class = cParmOp) or
124                         (nameTable (j).class = cChoiceOp)) and
125                         (nameTable (j).value = opCode) then
126                         found := true;
127                         index := j;  { Place in NameTable }
128                         opClass := nameTable (j).class;
129                     else
130                         j := j + 1;
131                     end if;
132                 end loop;
133
134                 if not found then
135                     index := notFound;  { 0 }
136                 end if;
137
138             end FindSemanticString;
139
140
141         procedure PrintSet (index: 0 .. maxSets,
142                             var i: 0 .. maxTableSize) =
143
144             imports (setTable, WriteString, ConvertNumber, maxNames,
145                      WriteLine, setSize, SetType, EmitCode, SetIn,
146                      blanks, flagValue, maxBits, LocateString,
147                      NameClasses);
148
149             { This procedure prints a set if it is not a duplicate;
150               it first emits the PowerSet values, and then each token
151               name in the set is printed }
152
153             begin
154                 var wordNo:
155                     0 .. setSize := 0;
156                 var temp:
157                     SetType;
158                 var printed:
159                     SignedInt := 0;
160                 var bitNo:
161                     SignedInt := 0;
162                 var bit:
163                     0 .. maxBits - 1;
164                 var indexx:
165                     0 .. maxNames;
166
167
168                 if not setTable (index).duplicate then
169                     WriteString ('    {  ');
170                     { Print index }
171                     ConvertNumber (setTable (index).root);
172                     WriteString ('   }     ');
```

```
173                     temp := setTable (index).sett;
174
175                     loop
176                         { Print powerset }
177                         ConvertNumber (temp (wordNo));
178                         WriteString (', ');
179                         exit when (wordNo = setSize);
180
181                         wordNo := wordNo + 1;  { Next subSet }
182                     end loop;
183
184                     WriteLine;
185                     WriteString (blanks);
186                     WriteString ('{ ');
187
188                     { Print token names in set }
189
190                     loop
191                         exit when (bitNo > totalBits);
192
193                         wordNo := bitNo div maxBits;
194                         bit := bitNo mod maxBits;
195                         if SetIn (temp (wordNo), flagValue (bit)) then
196                             { Print token name }
197                             LocateString (bitNo, cInput, indexx);
198                             EmitCode (indexx);
199                             printed := printed + 1;
200                             if printed mod 4 = 0 then
201                                 WriteLine;
202                                 WriteString (blanks);
203                             end if;
204                         end if;
205                         bitNo := bitNo + 1;
206                     end loop;
207
208                     WriteString (' }');
209                     WriteLine;
210                     i := i + 8;  { Number of table places for each set }
211
212                 end if;
213
214             end PrintSet;
215
216
217         procedure EmitSetValues (var i: 0 .. maxTableSize) =
218
219             imports (setTable, PrintSet, setTop, maxSets);
220
221             { This procedure processes each entry in the setTable};
222
223             begin
224                 var index:
```

```
225                     0 .. maxSets := 1;
226
227                 loop
228                     PrintSet (index, i);  { Prints each set in table }
229                     exit when (index = setTop);
230
231                     index := index + 1;
232                 end loop;
233
234             end EmitSetValues;
235
236
237         procedure EmitProcedures =
238
239             imports (var IO, tableFile, var tableAddress, EmitFixup,
240                  maxCalls, nameTable, var nameIndex, nameTop,
241                 ConvertNumber, maxNames, NameClasses, EmitCode,
242                 FindSemanticString, ProcessOperands, EmitOpCode,
243                 CheckForRule, ProcessInputChoiceTable, outputFile,
244                 inputChoiceTableTop, opCodeTable, AssemCode,
245                 WriteLine, WriteString, Error, EmitSetValues,
246                 noPredefinedNames);
247
248
249             { Resolves Procedure Calls and Emits the Assembled Table }
250
251             begin
252                 var i:
253                     0 .. maxTableSize;
254                 var j:
255                     1 .. maxCalls+1;
256                 var opCode:
257                     SignedInt;
258                 var index:
259                     0 .. maxNames; { location in nametable }
260                 var opClass:
261                     NameClasses; { opclass from nametable }
262                 var tableBegins:
263                     Boolean;  { check for beginning of
264                                 inputChoiceTable }
265
266                 IO.Open (tableFile, inFile);
267
268                 { Check for Unresolved Procedures }
269
270                 nameIndex := noPredefinedNames;
271                 loop
272                     exit when nameIndex = nameTop;
273                     nameIndex := nameIndex + 1;
274
275                     if nameTable(nameIndex).class = cProcedure and
276                             nameTable(nameIndex).value = nullValue
```

```
277                         then
278                         Error (eUndefinedProcedure);
279                     end if;
280                 end loop;
281
282                 { Emit Syntax/Semantic Table }
283
284                 WriteLine;
285                 IO.WriteChar (outputFile, tab);
286                 WriteString ('const tableSize := ');
287                 ConvertNumber (tableAddress);
288                 WriteString ('; '); {rmb}
289                 WriteLine;
290
291                 IO.WriteChar (outputFile, tab);
292                 WriteString ('const table:');
293                 WriteLine;
294                 IO.WriteChar (outputFile, tab);
295                 WriteString ('    array 0 .. tableSize of SignedInt
296                                                         := (');
297                 WriteLine;
298                 WriteLine;
299                 WriteString
300                     ('    { LOCATION     CONTENTS     RULE                ')
301                 WriteString
302                     ('ASSEMBLER CODE   }');
303                 WriteLine;
304                 WriteLine;
305
306                 j := 1;
307                 i := 0;
308
309                 loop  { through s-instructions }
310                     exit when i = saveTableAddress;
311
312                     { check for beginning of choice table }
313
314                     if (inputChoiceTableTop > 0) and
315                        (i = choiceTable (inputChoiceTableTop)) then
316                         tableBegins := true;
317                     else
318                         tableBegins := false;
319                     end if;
320
321                     WriteString ('    {  ');
322                     ConvertNumber (i); { Print index }
323                     WriteString ('   }     ');
324
325                     IO.ReadInt (tableFile, opCode);
326                     ConvertNumber (opCode); { Print opCode }
327                     WriteString (',  { ');
328
```

```
329                     { Name printed at start of new rule }
330
331                     if i = 0 then
332                         WriteString ('PROGRAM              ');
333                     else
334                         CheckForRule (i);
335                     end if;
336
337                     if tableBegins then
338                         ProcessInputChoiceTable (i, j, opCode);
339                         inputChoiceTableTop := inputChoiceTableTop - 1;
340                     else
341                         if opCode <= 8 then { primitive table
342                                                 operators }
343
344                             { Print opCode string }
345
346                             EmitOpCode (opCodeTable (opCode));
347                             WriteString (' }');
348                             WriteLine;
349
350                             case opCode of { process operands }
351
352                                 oCall, oRepeat, oMerge, oInput,
353                                     oEmit, oError, oInputChoice =>
354
355                                     ProcessOperands (opCode, i, j);
356                                     end oCall;
357
358                                 oReturn, oChoiceEnd =>
359                                     { no operands }
360                                     end oReturn;
361
362                             end case;
363
364                         else  { semantic operators }
365                             FindSemanticString (opCode, index,
366                                                 opClass);
367                             EmitCode (index);  { print opcode string }
368                             WriteString (' }');
369                             WriteLine;
370
371                             case opClass of
372                                 cSimpleOp =>
373                                 end cSimpleOp;
374
375                                 cParmOp =>
376                                 end cParmOp;
377
378                                 cChoiceOp =>
379                                 end cChoiceOp;
380
```

```
381                             cEmitOp =>   { no operands }
382                             end cEmitOp;
383                         end case;
384
385                     end if;
386
387                 end if;
388
389                 i := i + 1;  { next place in table }
390
391             end loop;
392
393             EmitSetValues (i);
394             WriteString ('    {  ');
395             ConvertNumber (i);
396             WriteString ('   }         0);');
397             WriteLine;
398
399             tableAddress := i;
400             IO.Close (tableFile);
401             IO.WriteChar (outputFile, endOfFile);
402
403         end EmitProcedures;
404
405
406
407     procedure AllocateSetSpace (setLabel: 0 .. maxSets) =
408
409         imports (var setTable, var tableAddress);
410
411         { This procedure allocates table space for each error
412           recovery set }
413         begin
414
415             setTable (setLabel).root := tableAddress;
416             { 8 table slots for each set }
417             tableAddress := tableAddress + 8;
418
419         end AllocateSetSpace;
420
421
422     procedure TestForDuplicate (var hashIndex: 0 .. hashSize,
423                                 setLabel: 0 .. maxSets,
424                                 var duplicate: Boolean) =
425
426         imports (setTable, setSize, hashTable, hashSize);
427
428         { This procedure determines if a set being processed is a
429           duplicate of a set already in the hashTable; if it is
430           not, an empty slot is returned }
431
432         begin
```

```
433                 var count:
434                     0 .. setSize;
435                 var found:
436                     Boolean := false;  { Empty slot or duplicate set }
437
438
439                 loop
440                     exit when (found);
441
442                     count := 0;
443                     duplicate := true;
444
445                     loop
446                         if setTable (hashTable (hashIndex)).sett
447                                     (count) not =
448                             setTable (setLabel).sett (count) then
449                             duplicate := false;
450                         end if;
451                         exit when (not duplicate) or (count = setSize);
452
453                         count := count + 1;
454                     end loop;
455
456                     if not duplicate then
457                         hashIndex := (hashIndex + 1) mod hashSize;
458                         if hashTable (hashIndex) = 0 then
459                             found := true;  { Empty slot }
460                         end if;
461                     else
462                         found := true;  { Duplicate set }
463                     end if;
464                 end loop;
465
466             end TestForDuplicate;
467
468
469
470
471         procedure HashSet (var node: SetTableEntry,
472                            setlabel: 0 .. maxSets) =
473
474             imports (var hashTable, var setTable, slotsFilled,
475                      AllocateSetSpace, hashSize, setSize,
476                      TestForDuplicate);
477
478             { This procedure hashes each set, checks for equality in
479               the case of identical hashes, and eliminates duplicate
480               sets }
481
482             begin
483                 var hashIndex:
484                     0 .. hashSize;
```

```
485                 var setTotal:
486                     SignedInt := 0;
487                 var i:
488                     0 .. setSize := 0;
489                 var duplication:
490                     Boolean;
491
492                 node.duplicate := false;
493
494                 { Addition of Powersets }
495
496                 loop
497                     setTotal := setTotal + (node.sett (i) mod
498                                                         hashSize);
499                     setTotal := setTotal mod hashSize;
500                     exit when (i = setSize);
501
502                     i := i + 1;
503                 end loop;
504
505                 hashIndex := setTotal;
506                 if hashTable (hashIndex) = 0 then  { Slot empty }
507                     hashTable (hashIndex) := setLabel;
508                     setTable (setLabel).duplicate := false;
509                     AllocateSetSpace (setLabel);
510                     slotsFilled := slotsFilled + 1;
511                 else  { Slot filled }
512                     TestForDuplicate (hashIndex, setLabel,
513                                       duplication);
514                     if not duplication then
515                     { Set hashed to same slot but not equal }
516                         hashTable (hashIndex) := setLabel;
517                         setTable (setLabel).duplicate := false;
518                         AllocateSetSpace (setLabel);
519                         slotsFilled := slotsFilled + 1;
520                     else  { Duplicate set }
521                         setTable (setLabel).duplicate := true;
522                         setTable (setLabel).root := setTable
523                                         (hashTable(hashIndex)).root;
524                     end if;
525                 end if;
526
527             end HashSet;
528
529
530         procedure EliminateDuplicateSets =
531
532             imports (var setTable, setTop, maxSets, slotsFilled,
533                      tableFull, HashSet);
534
535             { This procedure processes each set in the setTable for
536               possible duplication }
```

```
537
538             begin
539                 var setLabel:
540                     0 .. maxSets := 1;
541
542                 loop
543                     HashSet (setTable (setLabel),setLabel);
544                     exit when (setLabel = setTop);
545
546                     if slotsFilled > tableFull then
547                         Error (eHashTableTooFull);
548                     end if;
549                     setLabel := setLabel + 1;
550                 end loop;
551
552             end EliminateDuplicateSets;
553
554
555         initially
556
557             imports (var IO, inputFile, Abort, predefinedNameLength,
558                 noReservedErrorCodes, maxNames, noPredefinedNames,
559                 var token, var tokenText, var tokenLength,
560                 VerifyAndEnterName, var nameTable, var nameChars,
561                 errors, var listing, ProcessDefinitions,
562                 ProcessProcedures, EmitProcedures, var nextvalue,
563                 EliminateDuplicateSets, hashSize, predefinedNameChars,
564                 noPredefinedNameChars, outputFile, var lowerCase,
565                 var nextChar, var definitionClass, EmitDefinitions,
566                 CloseSets);
567
568
569             begin
570                 var i:
571                     1 .. noPredefinedNames;
572                 var j:
573                     1 .. noPredefinedNameChars+1;
574                 var k:
575                     1 .. maxNameLength;
576                 var c:
577                     ordCharFirst .. ordCharLast;
578
579                 const maxOptionStringLength := 100;
580                 var optionString:
581                     packed array 1..maxOptionStringLength of Char;
582                 var optionStringLength:
583                     SignedInt;
584                 var index:
585                     0 .. hashSize := 0;
586
587                 { Process Options }
588
```

```
589                 IO.GetOptions (optionString, optionStringLength);
590
591                 if optionStringLength > 0 then
592                     if optionString(1) = $L or optionString(1) = $l
593                         then
594                         listing := true;
595                     end if;
596                 end if;
597
598
599                 { Initialize Letter Normalization Map }
600                 { The following works for both ASCII and EBCDIC }
601
602                 c := ordCharFirst;
603                 loop
604                     lowerCase(c) := Chr(c);
605                     exit when c = ordCharLast;
606                     c := c + 1;
607                 end loop;
608
609                 lowerCase(Char.Ord($A)) := $a;
610                 lowerCase(Char.Ord($B)) := $b;
611                 lowerCase(Char.Ord($C)) := $c;
612                 lowerCase(Char.Ord($D)) := $d;
613                 lowerCase(Char.Ord($E)) := $e;
614                 lowerCase(Char.Ord($F)) := $f;
615                 lowerCase(Char.Ord($G)) := $g;
616                 lowerCase(Char.Ord($H)) := $h;
617                 lowerCase(Char.Ord($I)) := $i;
618                 lowerCase(Char.Ord($J)) := $j;
619                 lowerCase(Char.Ord($K)) := $k;
620                 lowerCase(Char.Ord($L)) := $l;
621                 lowerCase(Char.Ord($M)) := $m;
622                 lowerCase(Char.Ord($N)) := $n;
623                 lowerCase(Char.Ord($O)) := $o;
624                 lowerCase(Char.Ord($P)) := $p;
625                 lowerCase(Char.Ord($Q)) := $q;
626                 lowerCase(Char.Ord($R)) := $r;
627                 lowerCase(Char.Ord($S)) := $s;
628                 lowerCase(Char.Ord($T)) := $t;
629                 lowerCase(Char.Ord($U)) := $u;
630                 lowerCase(Char.Ord($V)) := $v;
631                 lowerCase(Char.Ord($W)) := $w;
632                 lowerCase(Char.Ord($X)) := $x;
633                 lowerCase(Char.Ord($Y)) := $y;
634                 lowerCase(Char.Ord($Z)) := $z;
635
636                 { Initialize Name Table }
637
638                 assert (none = 0);
639
640                 nameChars(none) := blank;
```

```
641
642                     assert (notFound = 0);
643
644                     nameTable(notFound).class := cValue;
645                     nameTable(notFound).charsIndex := none;
646                     nameTable(notFound).length := 1;
647                     nameTable(notFound).value := nullValue;
648
649                     assert (maxNames >= noPredefinedNames);
650
651                     token := tName;
652                     definitionClass := cClass;
653                     i := 1;
654                     j := 1;
655
656                     loop
657                         tokenLength := predefinedNameLength(i);
658
659                         k := 1;
660                         loop
661                             tokenText(k) := predefinedNameChars(j);
662                             j := j + 1;
663
664                             exit when k = tokenLength;
665                             k := k + 1;
666                         end loop;
667
668                         VerifyAndEnterName;
669
670                         exit when i = noPredefinedNames;
671                         i := i + 1;
672                     end loop;
673
674                     { Initialize the next value for each class of name }
675
676                     nextValue(cInput) := 0;
677                     nextValue(cOutput) := 0;
678                     nextValue(cError) := noReservedErrorCodes + 1;
679                     nextValue(cSimpleOp) := lastPrimitiveOperation + 1;
680                     nextValue(cParmOp) := lastPrimitiveOperation + 1;
681                     nextValue(cChoiceOp) := lastPrimitiveOperation + 1;
682                     nextValue(cEmitOp) := lastPrimitiveOperation + 1;
683                     nextValue(cValue) := 0;
684
685                     { Initialize Input }
686
687                     IO.Open (inputFile, inFile);
688                     IO.Open (outputFile, outFile);
689                     nextChar := newLine;
690
691
692                     { Initialize opCodeTable with primitive operations for
```

```
693                   printing Table }
694
695                opCodeTable (0) := 'Call       ';
696                opCodeTable (1) := 'Return     ';
697                opCodeTable (2) := 'Repeat     ';
698                opCodeTable (3) := 'Merge      ';
699                opCodeTable (4) := 'Input      ';
700                opCodeTable (5) := 'InputChoice';
701                opCodeTable (6) := 'Emit       ';
702                opCodeTable (7) := 'Error      ';
703                opCodeTable (8) := 'ChoiceEnd  ';
704
705                { Initialize hashTable for duplicate set elimination }
706
707                loop
708                    hashTable (index) := 0;
709                    exit when (index = hashSize);
710
711                    index := index + 1;
712                end loop;
713
714                { Process Definition Section }
715                ProcessDefinitions;
716                EmitDefinitions;
717
718                { Process Procedure Section }
719
720                ProcessProcedures;
721                CloseSets;
722                EliminateDuplicateSets;
723                EmitProcedures;
724
725
726                { Terminate Listing }
727
728                if listing then
729                    IO.WriteChar (outputFile, nextChar);
730                end if;
731
732                { Terminate Input }
733
734                IO.Close (inputFile);
735                IO.Close (outputFile);
736
737
738                { Indicate Errors by Abort }
739
740                if errors then
741                    Abort;
742                end if;
743
744            end;
```

```
745
746         end module; {Assembler}
747
748     include 'IOEND'
```

```
1   include 'IOSTART'
2   include 'POWERSET'
3
4   var Parser:
5       module
6
7           imports (var IO, Abort);
8
9
10          {    Project Euclid
11
12              Module: Euclid Parser V2.10
13              Author: J.R. Cordy
14              Reader: R.C. Holt
15              Date:   29 May 1979                }
16
17
18          {   This module is the interpreter for Euclid
19           Syntax/Semantic Language programs.  The Euclid
20           Parser will consist of this module plus the Syntax
21           Language representation of the syntax of Euclid.
22           The Semantic Passes will each consist of this module
23           plus the Syntax/Semantic Language representation
24           of the semantic processing to be done and the
25           Semantic Mechanisms module for the pass, which
26           implements the semantic operations required for
27           the pass.  The Transliterator Parser is a hybrid
28           and will contain some semantic operations
29           required to perform the transliteration.
30
31              A Syntax/Semantic Language (S/SL) program must be
32           processed by the S/SL Assembler, which will output
33           Euclid declarations for the Syntax/Semantic table and
34           the constants defining the table operation codes,
35           input tokens, output tokens, error codes and
36           semantic operation codes used in the table.
37
38
39           Automatic Generation of Syntax Error Recovery
40           Grace Evans, Summer 1982 }
41
42
43          { Primitive Table Operations:
44
45           These will remain the same independent of the
46           pass and form the fundamental table operations. }
47
48          pervasive const firstTableOperation := 0;
49          pervasive const firstPrimitiveOperation := 0;
50
51          pervasive const oCall := 0;
52          pervasive const oReturn := 1;
```

```
53          pervasive const oRepeat := 2;
54          pervasive const oMerge := 3;
55          pervasive const oInput := 4;
56          pervasive const oInputChoice := 5;
57          pervasive const oEmit := 6;
58          pervasive const oError := 7;
59          pervasive const oChoiceEnd := 8;
60
61          pervasive const lastPrimitiveOperation := 8;
62
63
64          { Semantic Operations:
65
66           These will be different for each pass.  The
67           semantic operations are implemented in the
68           Semantic module for the pass.            }
69
70          pervasive const firstSemanticOperation := 9;
71
72          pervasive const firstSimpleOperation := 9;
73          pervasive const lastSimpleOperation := 9;
74
75          pervasive const firstParameterizedOperation := 9;
76          pervasive const lastParameterizedOperation := 9;
77
78          pervasive const firstChoiceOperation := 9;
79          pervasive const lastChoiceOperation := 9;
80
81          pervasive const firstEmittingOperation := 9;
82          pervasive const oEmitIdent := 9;
83          pervasive const oEmitChar := 10;
84          pervasive const oEmitNumber := 11;
85          pervasive const oEmitString := 12;
86          pervasive const oEmitCode := 13;
87          pervasive const oEmitLine := 14;
88          pervasive const lastEmittingOperation := 14;
89
90          pervasive const lastSemanticOperation :=
91                                                  lastEmittingOperation;
92          pervasive const lastTableOperation := lastSemanticOperation;
93
94
95
96      { The Syntax/Semantic Table }
97
98      const tableSize :=  5010;
99      const table:
100         array 0 .. tableSize of SignedInt := (
101
102       { LOCATION     CONTENTS     RULE                ASSEMBLER CODE    }
103
104       {      0   }        14,  { PROGRAM               oEmitLine        }
```

```
105     {    1  }     0,  {                    Call          }
106                  287,  {                    Pervasive     }
107                 3538,  {                                  }
108     {    4  }     4,  {                    Input         }
109                  103,  {                    tType         }
110                  268,
111                 3546,  {                                  }
112     {    8  }     0,  {                    Call          }
113                  964,  {                    TypeDeclaration }
114                 3562,  {                                  }
115     {   11  }     4,  {                    Input         }
116                   35,  {                    tSemicolon    }
117                  268,
118                 3570,  {                                  }
119     {   15  }     6,  {                    Emit          }
120                   10,  {                    aEndOfFile    }
121     {   17  }     1,  {                    Return        }
122     {   18  }     5,  { ModuleBody         InputChoice   }
123                   29,  {                    TABLE         }
124                  276,
125                 3586,  {                                  }
126     {   22  }     6,  {                    Emit          }
127                   91,  {                    aModuleIdent  }
128     {   24  }     6,  {                    Emit          }
129                    0,  {                    aIdent        }
130     {   26  }     9,  {                    oBmutIdent    }
131     {   27  }     3,  {                    Merge         }
132                   32,  {                                  }
133     {   29  }     1,  {                    TABLE BEGINS  }
134                    0,  {                    tIdent        }
135                   22,
136     {   32  }     4,  {                    Input         }
137                   18,  {                    tNewLine      }
138                  281,
139                 3594,  {                                  }
140     {   36  }    14,  {                    oBmitLine     }
141     {   37  }     0,  {                    Call          }
142                  183,  {                    ImportClauses }
143                 3594,  {                                  }
144     {   40  }     0,  {                    Call          }
145                  354,  {                    ExportClause  }
146                 3618,  {                                  }
147     {   43  }     5,  {                    InputChoice   }
148                   69,  {                    TABLE         }
149                  284,
150                 3618,  {                                  }
151     {   47  }     4,  {                    Input         }
152                   53,  {                    tChecked      }
153                  286,
154                 3626,  {                                  }
155     {   51  }     6,  {                    Emit          }
156                   94,  {                    aNotChecked   }
```

```
157   {  53 }     4,  {   Input              }
158              35,  {   tSemicolon         }
159             286,
160            3626,  {                      }
161   {  57 }    14,  {   oEmitLine          }
162   {  58 }     3,  {   Merge              }
163              74,  {                      }
164   {  60 }     6,  {   Emit               }
165              27,  {   aChecked           }
166   {  62 }     4,  {   Input              }
167              35,  {   tSemicolon         }
168             289,
169            3626,  {                      }
170   {  66 }    14,  {   oEmitLine          }
171   {  67 }     3,  {   Merge              }
172              74,  {                      }
173   {  69 }     2,  {   TABLE BEGINS       }
174              85,  {   tNot               }
175              47,
176              53,  {   tChecked           }
177              60,
178   {  74 }     0,  {   Call               }
179             488,  {   Declarations       }
180            3642,  {                      }
181   {  77 }     5,  {   InputChoice        }
182              93,  {   TABLE              }
183             294,
184            3642,  {                      }
185   {  81 }     6,  {   Emit               }
186              79,  {   aInitially         }
187   {  83 }     0,  {   Call               }
188            2161,  {   RoutineDefiniti    }
189            3658,  {                      }
190   {  86 }     4,  {   Input              }
191              35,  {   tSemicolon         }
192             296,
193            3658,  {                      }
194   {  90 }    14,  {   oEmitLine          }
195   {  91 }     3,  {   Merge              }
196              96,  {                      }
197   {  93 }     1,  {   TABLE BEGINS       }
198              78,  {   tInitially         }
199              81,
200   {  96 }     5,  {   InputChoice        }
201             158,  {   TABLE              }
202             300,
203            3658,  {                      }
204   { 100 }     6,  {   Emit               }
205              11,  {   aAbstraction       }
206   { 102 }     4,  {   Input              }
207              73,  {   tFunction          }
208             302,
```

```
209                          3666,  {                          }
210        {   106   }         0,  {   Call                   }
211                          2030,  {   FunctionDeclara        }
212                          3666,  {                          }
213        {   109   }         4,  {   Input                  }
214                            35,  {   tSemicolon             }
215                           302,
216                          3666,  {                          }
217        {   113   }         4,  {   Input                  }
218                            80,  {   tInvariant             }
219                           303,
220                          3674,  {                          }
221        {   117   }        14,  {   oEmitLine              }
222        {   118   }         6,  {   Emit                   }
223                            81,  {   aInvariant             }
224        {   120   }         4,  {   Input                  }
225                            29,  {   tLeftParen             }
226                           303,
227                          3674,  {                          }
228        {   124   }         0,  {   Call                   }
229                          3000,  {   Expression             }
230                          3690,  {                          }
231        {   127   }         4,  {   Input                  }
232                            30,  {   tRightParen            }
233                           303,
234                          3698,  {                          }
235        {   131   }         4,  {   Input                  }
236                            35,  {   tSemicolon             }
237                           303,
238                          3698,  {                          }
239        {   135   }        14,  {   oEmitLine              }
240        {   136   }         3,  {   Merge                  }
241                           163,  {                          }
242        {   138   }         6,  {   Emit                   }
243                            81,  {   aInvariant             }
244        {   140   }         4,  {   Input                  }
245                            29,  {   tLeftParen             }
246                           306,
247                          3674,  {                          }
248        {   144   }         0,  {   Call                   }
249                          3000,  {   Expression             }
250                          3690,  {                          }
251        {   147   }         4,  {   Input                  }
252                            30,  {   tRightParen            }
253                           306,
254                          3698,  {                          }
255        {   151   }         4,  {   Input                  }
256                            35,  {   tSemicolon             }
257                           306,
258                          3698,  {                          }
259        {   155   }        14,  {   oEmitLine              }
260        {   156   }         3,  {   Merge                  }
```

```
261                          163,  {                                 }
262         {   158  }         2,  {  TABLE BEGINS                   }
263                           39,  {  tAbstraction                   }
264                          100,
265                           80,  {  tInvariant                     }
266                          138,
267         {   163  }         5,  {  InputChoice                    }
268                          179,  {  TABLE                         }
269                          310,
270                         3698,  {                                 }
271         {   167  }         6,  {  Emit                          }
272                           64,  {  aFinally                       }
273         {   169  }         0,  {  Call                          }
274                         2161,  {  RoutineDefiniti                }
275                         3562,  {                                 }
276         {   172  }         4,  {  Input                         }
277                           35,  {  tSemicolon                     }
278                          312,
279                         3570,  {                                 }
280         {   176  }        14,  {  oEmitLine                     }
281         {   177  }         3,  {  Merge                         }
282                          182,  {                                 }
283         {   179  }         1,  {  TABLE BEGINS                  }
284                           69,  {  tFinally                       }
285                          167,
286         {   182  }         1,  {  Return                        }
287         {   183  }         5,  {  ImportClauses  InputChoice    }
288                          265,  {  TABLE                         }
289                          321,
290                         3706,  {                                 }
291         {   187  }         6,  {  Emit                          }
292                           72,  {  aImports                       }
293         {   189  }         4,  {  Input                         }
294                           29,  {  tLeftParen                     }
295                          323,
296                         3706,  {                                 }
297         {   193  }         0,  {  Call                          }
298                          273,  {  ImportItem                     }
299                         3706,  {                                 }
300         {   196  }         5,  {  InputChoice                   }
301                          209,  {  TABLE                         }
302                          324,
303                         3706,  {                                 }
304         {   200  }         3,  {  Merge                         }
305                          218,  {                                 }
306         {   202  }         3,  {  Merge                         }
307                          216,  {                                 }
308         {   204  }         0,  {  Call                          }
309                          273,  {  ImportItem                     }
310                         3706,  {                                 }
311         {   207  }         3,  {  Merge                         }
312                          216,  {                                 }
```

```
313         {    209   }        2,  {                TABLE BEGINS    }
314                              30,  {               tRightParen     }
315                             200,
316                              34,  {               tComma          }
317                             204,
318         {    214   }        2,  {                Repeat          }
319                             200,  {                              }
320         {    216   }        2,  {                Repeat          }
321                             196,  {                              }
322         {    218   }        6,  {                Emit            }
323                              48,  {               aEndImports     }
324         {    220   }        5,  {                InputChoice     }
325                             255,  {               TABLE           }
326                             331,
327                            3706,  {                              }
328         {    224   }        4,  {                Input           }
329                              29,  {               tLeftParen      }
330                             333,
331                            3706,  {                              }
332         {    228   }        0,  {                Call            }
333                             323,  {               ThusItem        }
334                            3706,  {                              }
335         {    231   }        5,  {                InputChoice     }
336                             244,  {               TABLE           }
337                             334,
338                            3706,  {                              }
339         {    235   }        3,  {                Merge           }
340                             253,  {                              }
341         {    237   }        3,  {                Merge           }
342                             251,  {                              }
343         {    239   }        0,  {                Call            }
344                             323,  {               ThusItem        }
345                            3706,  {                              }
346         {    242   }        3,  {                Merge           }
347                             251,  {                              }
348         {    244   }        2,  {                TABLE BEGINS    }
349                              30,  {               tRightParen     }
350                             235,
351                              34,  {               tComma          }
352                             239,
353         {    249   }        2,  {                Repeat          }
354                             235,  {                              }
355         {    251   }        2,  {                Repeat          }
356                             231,  {                              }
357         {    253   }        3,  {                Merge           }
358                             258,  {                              }
359         {    255   }        1,  {                TABLE BEGINS    }
360                             101,  {               tThus           }
361                             224,
362         {    258   }        4,  {                Input           }
363                              35,  {               tSemicolon      }
364                             342,
```

```
365                  3706,  {                                     }
366   {  262  }        14,  {                     oEmitLine       }
367   {  263  }         3,  {                     Merge           }
368                   270,  {                                     }
369   {  265  }         1,  {                     TABLE BEGINS    }
370                    75,  {                     tImports        }
371                   187,
372   {  268  }         3,  {                     Merge           }
373                   272,  {                                     }
374   {  270  }         2,  {                     Repeat          }
375                   183,  {                                     }
376   {  272  }         1,  {                     Return          }
377   {  273  }         0,  { ImportItem          Call            }
378                   287,  {                     Pervasive       }
379                  3722,  {                                     }
380   {  276  }         0,  {                     Call            }
381                   299,  {                     BindingConditio }
382                  3554,  {                                     }
383   {  279  }         4,  {                     Input           }
384                     0,  {                     tIdent          }
385                   351,
386                  3570,  {                                     }
387   {  283  }         6,  {                     Emit            }
388                     0,  {                     aIdent          }
389   {  285  }         9,  {                     oEmitIdent      }
390   {  286  }         1,  {                     Return          }
391   {  287  }         5,  { Pervasive           InputChoice     }
392                   295,  {                     TABLE           }
393                   357,
394                  3530,  {                                     }
395   {  291  }         6,  {                     Emit            }
396                   104,  {                     aPervasive      }
397   {  293  }         3,  {                     Merge           }
398                   298,  {                                     }
399   {  295  }         1,  {                     TABLE BEGINS    }
400                    91,  {                     tPervasive      }
401                   291,
402   {  298  }         1,  {                     Return          }
403   {  299  }         5,  { BindingConditio     InputChoice     }
404                   315,  {                     TABLE           }
405                   367,
406                  3730,  {                                     }
407   {  303  }         6,  {                     Emit            }
408                   122,  {                     aVar            }
409   {  305  }         3,  {                     Merge           }
410                   322,  {                                     }
411   {  307  }         6,  {                     Emit            }
412                   109,  {                     aReadonly       }
413   {  309  }         3,  {                     Merge           }
414                   322,  {                                     }
415   {  311  }         6,  {                     Emit            }
416                    30,  {                     aConst          }
```

```
417         {    313   }         3,  {                        Merge          }
418                              322,  {                                       }
419         {    315   }         3,  {                        TABLE BEGINS   }
420                              105,  {                        tVar           }
421                              303,
422                               95,  {                        tReadonly      }
423                              307,
424                               56,  {                        tConst         }
425                              311,
426         {    322   }         1,  {                        Return         }
427         {    323   }         5,  { ThusItem               InputChoice    }
428                              329,  {                        TABLE          }
429                              381,
430                             3714,  {                                       }
431         {    327   }         3,  {                        Merge          }
432                              332,  {                                       }
433         {    329   }         1,  {                        TABLE BEGINS   }
434                               91,  {                        tPervasive     }
435                              327,
436         {    332   }         5,  {                        InputChoice    }
437                              342,  {                        TABLE          }
438                              385,
439                             3722,  {                                       }
440         {    336   }         3,  {                        Merge          }
441                              349,  {                                       }
442         {    338   }         3,  {                        Merge          }
443                              349,  {                                       }
444         {    340   }         3,  {                        Merge          }
445                              349,  {                                       }
446         {    342   }         3,  {                        TABLE BEGINS   }
447                              105,  {                        tVar           }
448                              336,
449                               95,  {                        tReadonly      }
450                              338,
451                               56,  {                        tConst         }
452                              340,
453         {    349   }         4,  {                        Input          }
454                                0,  {                        tIdent         }
455                              391,
456                             3570,  {                                       }
457         {    353   }         1,  {                        Return         }
458         {    354   }         5,  { ExportClause           InputChoice    }
459                              372,  {                        TABLE          }
460                              397,
461                             3738,  {                                       }
462         {    358   }         6,  {                        Emit           }
463                               61,  {                        aExports       }
464         {    360   }         0,  {                        Call           }
465                              376,  {                        ExportList     }
466                             3562,  {                                       }
467         {    363   }         6,  {                        Emit           }
468                               44,  {                        aEndExports    }
```

```
469     {   365 }     4, {                    Input           }
470                  35, {                    tSemicolon      }
471                 399,
472                3570, {                                    }
473     {   369 }    14, {                    oEmitLine       }
474     {   370 }     3, {                    Merge           }
475                  375, {                                    }
476     {   372 }     1, {                    TABLE BEGINS    }
477                   67, {                    tExports        }
478                  358,
479     {   375 }     1, {                    Return          }
480     {   376 }     4, { ExportList         Input           }
481                   29, {                    tLeftParen      }
482                  408,
483                 3754, {                                    }
484     {   380 }     0, {                    Call            }
485                  406, {                    ExportItem      }
486                 3754, {                                    }
487     {   383 }     5, {                    InputChoice     }
488                  396, {                    TABLE           }
489                  409,
490                 3754, {                                    }
491     {   387 }     3, {                    Merge           }
492                  405, {                                    }
493     {   389 }     3, {                    Merge           }
494                  403, {                                    }
495     {   391 }     0, {                    Call            }
496                  406, {                    ExportItem      }
497                 3754, {                                    }
498     {   394 }     3, {                    Merge           }
499                  403, {                                    }
500     {   396 }     2, {                    TABLE BEGINS    }
501                   30, {                    tRightParen     }
502                  387,
503                   34, {                    tComma          }
504                  391,
505     {   401 }     2, {                    Repeat          }
506                  387, {                                    }
507     {   403 }     2, {                    Repeat          }
508                  383, {                                    }
509     {   405 }     1, {                    Return          }
510     {   406 }     5, { ExportItem         InputChoice     }
511                  448, {                    TABLE           }
512                  420,
513                 3770, {                                    }
514     {   410 }     6, {                    Emit            }
515                   19, {                    aAssign         }
516     {   412 }     3, {                    Merge           }
517                  470, {                                    }
518     {   414 }     6, {                    Emit            }
519                   59, {                    aEqual          }
520     {   416 }     3, {                    Merge           }
```

```
521                          470,  {                          }
522     {   418  }             6,  {  Emit                    }
523                          105,  {  aPointer                 }
524     {   420  }             3,  {  Merge                   }
525                          470,  {                          }
526     {   422  }             6,  {  Emit                    }
527                            0,  {  aIdent                  }
528     {   424  }             9,  {  oEmitIdent              }
529     {   425  }             5,  {  InputChoice             }
530                          440,  {  TABLE                   }
531                          429,
532                         3778,  {                          }
533     {   429  }             6,  {  Emit                    }
534                          117,  {  aTo                      }
535     {   431  }             4,  {  Input                   }
536                            0,  {  tIdent                  }
537                          431,
538                         3570,  {                          }
539     {   435  }             6,  {  Emit                    }
540                            0,  {  aIdent                  }
541     {   437  }             9,  {  oEmitIdent              }
542     {   438  }             3,  {  Merge                   }
543                          446,  {                          }
544     {   440  }             1,  {  TABLE BEGINS            }
545                           32,  {  tDoublePeriod           }
546                          429,
547     {   443  }             0,  {  Call                    }
548                          471,  {  WithClause              }
549                         3570,  {                          }
550     {   446  }             3,  {  Merge                   }
551                          470,  {                          }
552     {   448  }             4,  {  TABLE BEGINS            }
553                           33,  {  tAssign                 }
554                          410,
555                           23,  {  tEqual                  }
556                          414,
557                           37,  {  tUpArrow                }
558                          418,
559                            0,  {  tIdent                  }
560                          422,
561     {   457  }             0,  {  Call                    }
562                          299,  {  BindingConditio         }
563                         3794,  {                          }
564     {   460  }             4,  {  Input                   }
565                            0,  {  tIdent                  }
566                          437,
567                         3786,  {                          }
568     {   464  }             6,  {  Emit                    }
569                            0,  {  aIdent                  }
570     {   466  }             9,  {  oEmitIdent              }
571     {   467  }             0,  {  Call                    }
572                          471,  {  WithClause              }
```

```
                    3570,  {                                           }
    {  470  }          1,  {                      Return               }
    {  471  }          5,  { WithClause           InputChoice          }
                     484,  {                      TABLE                }
                     444,
                    3802,  {                                           }
    {  475  }          6,  {                      Emit                 }
                     125,  {                      aWith                }
    {  477  }          0,  {                      Call                 }
                     376,  {                      ExportList           }
                    3570,  {                                           }
    {  480  }          6,  {                      Emit                 }
                      57,  {                      aEndWith             }
    {  482  }          3,  {                      Merge                }
                     487,  {                                           }
    {  484  }          1,  {                      TABLE BEGINS         }
                     107,  {                      tWith                }
                     475,
    {  487  }          1,  {                      Return               }
    {  488  }          5,  { Declarations         InputChoice          }
                     592,  {                      TABLE                }
                     455,
                    3810,  {                                           }
    {  492  }          0,  {                      Call                 }
                     629,  {                      VariableDeclara      }
                    3810,  {                                           }
    {  495  }          3,  {                      Merge                }
                     621,  {                                           }
    {  497  }          0,  {                      Call                 }
                     705,  {                      VariableBinding      }
                    3810,  {                                           }
    {  500  }          3,  {                      Merge                }
                     621,  {                                           }
    {  502  }          0,  {                      Call                 }
                    2503,  {                      AssertStatement      }
                    3810,  {                                           }
    {  505  }          3,  {                      Merge                }
                     621,  {                                           }
    {  507  }          6,  {                      Emit                 }
                     104,  {                      aPervasive           }
    {  509  }          5,  {                      InputChoice          }
                     543,  {                      TABLE                }
                     464,
                    3810,  {                                           }
    {  513  }          0,  {                      Call                 }
                     789,  {                      ConstantDeclara      }
                    3810,  {                                           }
    {  516  }          3,  {                      Merge                }
                     558,  {                                           }
    {  518  }          0,  {                      Call                 }
                     964,  {                      TypeDeclaration      }
                    3810,  {                                           }
```

```
{ 521 }     3,   { Merge                }
          558,   {                      }
{ 523 }     0,   { Call                 }
          907,   { ConverterDeclar      }
         3810,   {                      }
{ 526 }     3,   { Merge                }
          558,   {                      }
{ 528 }     0,   { Call                 }
         1116,   { InlineProcedure      }
         3810,   {                      }
{ 531 }     3,   { Merge                }
          558,   {                      }
{ 533 }     0,   { Call                 }
         2006,   { ProcedureDeclar      }
         3810,   {                      }
{ 536 }     3,   { Merge                }
          558,   {                      }
{ 538 }     0,   { Call                 }
         2030,   { FunctionDeclara      }
         3810,   {                      }
{ 541 }     3,   { Merge                }
          558,   {                      }
{ 543 }     6,   { TABLE BEGINS         }
           56,   { tConst               }
          513,
          103,   { tType                }
          518,
           57,   { tConverter           }
          523,
           79,   { tInline              }
          528,
           94,   { tProcedure           }
          533,
           73,   { tFunction            }
          538,
{ 556 }     2,   { Repeat               }
          513,   {                      }
{ 558 }     3,   { Merge                }
          621,   {                      }
{ 560 }     0,   { Call                 }
          789,   { ConstantDeclara      }
         3810,   {                      }
{ 563 }     3,   { Merge                }
          621,   {                      }
{ 565 }     0,   { Call                 }
          964,   { TypeDeclaration      }
         3810,   {                      }
{ 568 }     3,   { Merge                }
          621,   {                      }
{ 570 }     0,   { Call                 }
          907,   { ConverterDeclar      }
         3810,   {                      }
```

```
{ 573 }     3,  {   Merge              }
          621,  {                      }
{ 575 }     0,  {   Call               }
         1116,  {   InlineProcedure    }
         3810,  {                      }
{ 578 }     3,  {   Merge              }
          621,  {                      }
{ 580 }     0,  {   Call               }
         2006,  {   ProcedureDeclar    }
         3810,  {                      }
{ 583 }     3,  {   Merge              }
          621,  {                      }
{ 585 }     0,  {   Call               }
         2030,  {   FunctionDeclara    }
         3810,  {                      }
{ 588 }     3,  {   Merge              }
          621,  {                      }
{ 590 }     3,  {   Merge              }
          621,  {                      }
{ 592 }    11,  {   TABLE BEGINS       }
          105,  {   tVar               }
          492,
           48,  {   tBind              }
          497,
           45,  {   tAssert            }
          502,
           91,  {   tPervasive         }
          507,
           56,  {   tConst             }
          560,
          103,  {   tType              }
          565,
           57,  {   tConverter         }
          570,
           79,  {   tInline            }
          575,
           94,  {   tProcedure         }
          580,
           73,  {   tFunction          }
          585,
           35,  {   tSemicolon         }
          590,
{ 615 }     4,  {   Input              }
           35,  {   tSemicolon         }
          492,
         3562,  {                      }
{ 619 }     3,  {   Merge              }
          628,  {                      }
{ 621 }     4,  {   Input              }
           35,  {   tSemicolon         }
          494,
         3570,  {                      }
```

```
729         {    625   }        14,  {                        oEmitLine      }
730         {    626   }         2,  {                        Repeat         }
731                              488,  {                                       }
732         {    628   }         1,  {                        Return         }
733         {    629   }         6,  { VariableDeclara        Emit           }
734                              122,  {                        aVar           }
735         {    631   }         4,  {                        Input          }
736                                0,  {                        tIdent         }
737                              501,
738                             3834,  {                                       }
739         {    635   }         6,  {                        Emit           }
740                                0,  {                        aIdent         }
741         {    637   }         9,  {                        oEmitIdent     }
742         {    638   }         5,  {                        InputChoice    }
743                              657,  {                        TABLE          }
744                              502,
745                             3834,  {                                       }
746         {    642   }         4,  {                        Input          }
747                               46,  {                        tAt            }
748                              504,
749                             3842,  {                                       }
750         {    646   }         6,  {                        Emit           }
751                               20,  {                        aAt            }
752         {    648   }         0,  {                        Call           }
753                             3000,  {                        Expression     }
754                             3842,  {                                       }
755         {    651   }         4,  {                        Input          }
756                               30,  {                        tRightParen    }
757                              504,
758                             3850,  {                                       }
759         {    655   }         3,  {                        Merge          }
760                              680,  {                                       }
761         {    657   }         1,  {                        TABLE BEGINS   }
762                               29,  {                        tLeftParen     }
763                              642,
764         {    660   }         5,  {                        InputChoice    }
765                              673,  {                        TABLE          }
766                              506,
767                             3858,  {                                       }
768         {    664   }         4,  {                        Input          }
769                                0,  {                        tIdent         }
770                              508,
771                             3858,  {                                       }
772         {    668   }         6,  {                        Emit           }
773                                0,  {                        aIdent         }
774         {    670   }         9,  {                        oEmitIdent     }
775         {    671   }         3,  {                        Merge          }
776                              678,  {                                       }
777         {    673   }         1,  {                        TABLE BEGINS   }
778                               34,  {                        tComma         }
779                              664,
780         {    676   }         3,  {                        Merge          }
```

```
781                          680,  {                                }
782     {   678  }             2,  {  Repeat                         }
783                          660,  {                                }
784     {   680  }             4,  {  Input                          }
785                           36,  {  tColon                        }
786                          513,
787                         3866,  {                                }
788     {   684  }            14,  {  oEmitLine                     }
789     {   685  }             6,  {  Emit                          }
790                          123,  {  aVarType                      }
791     {   687  }             0,  {  Call                          }
792                         1140,  {  TypeDefinition                }
793                         3882,  {                                }
794     {   690  }             5,  {  InputChoice                   }
795                          701,  {  TABLE                         }
796                          514,
797                         3882,  {                                }
798     {   694  }             6,  {  Emit                          }
799                           78,  {  aInitial                      }
800     {   696  }             0,  {  Call                          }
801                         3000,  {  Expression                    }
802                         3570,  {                                }
803     {   699  }             3,  {  Merge                         }
804                          704,  {                                }
805     {   701  }             1,  {  TABLE BEGINS                  }
806                           33,  {  tAssign                       }
807                          694,
808     {   704  }             1,  {  Return                        }
809     {   705  }             6,  {  VariableBinding  Emit         }
810                           22,  {  aBind                         }
811     {   707  }             5,  {  InputChoice                   }
812                          738,  {  TABLE                         }
813                          525,
814                         3890,  {                                }
815     {   711  }             0,  {  Call                          }
816                          747,  {  RenameVariable                }
817                         3906,  {                                }
818     {   714  }             5,  {  InputChoice                   }
819                          727,  {  TABLE                         }
820                          528,
821                         3906,  {                                }
822     {   718  }             3,  {  Merge                         }
823                          736,  {                                }
824     {   720  }             3,  {  Merge                         }
825                          734,  {                                }
826     {   722  }             0,  {  Call                          }
827                          747,  {  RenameVariable                }
828                         3906,  {                                }
829     {   725  }             3,  {  Merge                         }
830                          734,  {                                }
831     {   727  }             2,  {  TABLE BEGINS                  }
832                           30,  {  tRightParen                   }
```

```
833                          718,
834                           34,  {                      tComma          }
835                          722,
836     {   732  }            2,  {                      Repeat          }
837                          718,  {                                      }
838     {   734  }            2,  {                      Repeat          }
839                          714,  {                                      }
840     {   736  }            3,  {                      Merge           }
841                          744,  {                                      }
842     {   738  }            1,  {                      TABLE BEGINS    }
843                           29,  {                      tLeftParen      }
844                          711,
845     {   741  }            0,  {                      Call            }
846                          747,  {                      RenameVariable  }
847                         3570,  {                                      }
848     {   744  }            6,  {                      Emit            }
849                           40,  {                      aEndBind        }
850     {   746  }            1,  {                      Return          }
851     {   747  }            0,  { RenameVariable       Call            }
852                          771,  {                      VarBindingCondi }
853                         3922,  {                                      }
854     {   750  }            4,  {                      Input           }
855                           0,  {                      tIdent          }
856                          543,
857                         3922,  {                                      }
858     {   754  }            6,  {                      Emit            }
859                           0,  {                      aIdent          }
860     {   756  }            9,  {                      oEmitIdent      }
861     {   757  }            4,  {                      Input           }
862                         102,  {                      tTo             }
863                          543,
864                         3930,  {                                      }
865     {   761  }            6,  {                      Emit            }
866                         117,  {                      aTo             }
867     {   763  }            4,  {                      Input           }
868                           0,  {                      tIdent          }
869                          543,
870                         3938,  {                                      }
871     {   767  }            0,  {                      Call            }
872                         3407,  {                      Variable        }
873                         3570,  {                                      }
874     {   770  }            1,  {                      Return          }
875     {   771  }            5,  { VarBindingCondi      InputChoice     }
876                          783,  {                      TABLE           }
877                          549,
878                         3914,  {                                      }
879     {   775  }            6,  {                      Emit            }
880                         122,  {                      aVar            }
881     {   777  }            3,  {                      Merge           }
882                          788,  {                                      }
883     {   779  }            6,  {                      Emit            }
884                         109,  {                      aReadonly       }
```

```
885     {   781 }     3,  {                    Merge          }
886                 788,  {                                   }
887     {   783 }     2,  {                    TABLE BEGINS   }
888                 105,  {                    tVar           }
889                 775,
890                  95,  {                    tReadonly      }
891                 779,
892     {   788 }     1,  {                    Return         }
893     {   789 }     6,  { ConstantDeclara    Emit           }
894                  30,  {                    aConst         }
895     {   791 }     4,  {                    Input          }
896                   0,  {                    tIdent         }
897                 561,
898                3946,  {                                   }
899     {   795 }     6,  {                    Emit           }
900                   0,  {                    aIdent         }
901     {   797 }     9,  {                    oEmitIdent     }
902     {   798 }     5,  {                    InputChoice    }
903                 811,  {                    TABLE          }
904                 562,
905                3946,  {                                   }
906     {   802 }     4,  {                    Input          }
907                   0,  {                    tIdent         }
908                 564,
909                3946,  {                                   }
910     {   806 }     6,  {                    Emit           }
911                   0,  {                    aIdent         }
912     {   808 }     9,  {                    oEmitIdent     }
913     {   809 }     3,  {                    Merge          }
914                 816,  {                                   }
915     {   811 }     1,  {                    TABLE BEGINS   }
916                  34,  {                    tComma         }
917                 802,
918     {   814 }     3,  {                    Merge          }
919                 818,  {                                   }
920     {   816 }     2,  {                    Repeat         }
921                 798,  {                                   }
922     {   818 }     5,  {                    InputChoice    }
923                 862,  {                    TABLE          }
924                 568,
925                3946,  {                                   }
926     {   822 }     6,  {                    Emit           }
927                  78,  {                    aInitial       }
928     {   824 }     0,  {                    Call           }
929                3000,  {                    Expression     }
930                3570,  {                                   }
931     {   827 }     3,  {                    Merge          }
932                 869,  {                                   }
933     {   829 }    14,  {                    oEmitLine      }
934     {   830 }     6,  {                    Emit           }
935                  31,  {                    aConstType     }
936     {   832 }     0,  {                    Call           }
```

```
937                          1140,  {                        TypeDefinition  }
938                          3954,  {                                        }
939     {    835    }          4,  {                        Input           }
940                            33,  {                        tAssign         }
941                           573,
942                          3962,  {                                        }
943     {    839    }          5,  {                        InputChoice     }
944                           852,  {                        TABLE           }
945                           574,
946                          3962,  {                                        }
947     {    843    }          0,  {                        Call            }
948                           870,  {                        StructuredConst }
949                          3978,  {                                        }
950     {    846    }          4,  {                        Input           }
951                            30,  {                        tRightParen     }
952                           576,
953                          3570,  {                                        }
954     {    850    }          3,  {                        Merge           }
955                           860,  {                                        }
956     {    852    }          1,  {                        TABLE BEGINS    }
957                            29,  {                        tLeftParen      }
958                           843,
959     {    855    }          6,  {                        Emit            }
960                            78,  {                        aInitial        }
961     {    857    }          0,  {                        Call            }
962                          3000,  {                        Expression      }
963                          3570,  {                                        }
964     {    860    }          3,  {                        Merge           }
965                           869,  {                                        }
966     {    862    }          2,  {                        TABLE BEGINS    }
967                            33,  {                        tAssign         }
968                           822,
969                            36,  {                        tColon          }
970                           829,
971     {    867    }          2,  {                        Repeat          }
972                           822,  {                                        }
973     {    869    }          1,  {                        Return          }
974     {    870    }          6,  { StructuredConst        Emit            }
975                           121,  {                        aValues         }
976     {    872    }          5,  {                        InputChoice     }
977                           885,  {                        TABLE           }
978                           588,
979                          3962,  {                                        }
980     {    876    }          0,  {                        Call            }
981                           870,  {                        StructuredConst }
982                          3962,  {                                        }
983     {    879    }          4,  {                        Input           }
984                            30,  {                        tRightParen     }
985                           590,
986                          3962,  {                                        }
987     {    883    }          3,  {                        Merge           }
988                           891,  {                                        }
```

```
989     {   885 }     1, {                    TABLE BEGINS    }
990                  29, {                    tLeftParen      }
991                 876,
992     {   888 }     0, {                    Call            }
993                3000, {                    Expression      }
994                3962, {                                    }
995     {   891 }     5, {                    InputChoice     }
996                 897, {                    TABLE           }
997                 594,
998                3986, {                                    }
999     {   895 }     3, {                    Merge           }
000                 902, {                                    }
001     {   897 }     1, {                    TABLE BEGINS    }
002                  34, {                    tComma          }
003                 895,
004     {   900 }     3, {                    Merge           }
005                 904, {                                    }
006     {   902 }     2, {                    Repeat          }
007                 872, {                                    }
008     {   904 }     6, {                    Emit            }
009                  56, {                    aEndValues      }
010     {   906 }     1, {                    Return          }
011     {   907 }     6, { ConverterDeclar    Emit            }
012                  32, {                    aConverter      }
013     {   909 }     4, {                    Input           }
014                   0, {                    tIdent          }
015                 606,
016                3994, {                                    }
017     {   913 }     6, {                    Emit            }
018                   0, {                    aIdent          }
019     {   915 }     9, {                    oEmitIdent      }
020     {   916 }     4, {                    Input           }
021                  29, {                    tLeftParen      }
022                 606,
023                4002, {                                    }
024     {   920 }     5, {                    InputChoice     }
025                 937, {                    TABLE           }
026                 607,
027                4002, {                                    }
028     {   924 }     6, {                    Emit            }
029                   0, {                    aIdent          }
030     {   926 }     9, {                    oEmitIdent      }
031     {   927 }     3, {                    Merge           }
032                 946, {                                    }
033     {   929 }     6, {                    Emit            }
034                 108, {                    aProcedure      }
035     {   931 }     3, {                    Merge           }
036                 946, {                                    }
037     {   933 }     6, {                    Emit            }
038                  67, {                    aFunction       }
039     {   935 }     3, {                    Merge           }
040                 946, {                                    }
```

```
041         {    937   }        3,  {                        TABLE BEGINS    }
042                               0,  {                        tIdent          }
043                             924,
044                              94,  {                        tProcedure      }
045                             929,
046                              73,  {                        tFunction       }
047                             933,
048         {    944   }        2,  {                        Repeat          }
049                             924,  {                                        }
050         {    946   }        4,  {                        Input           }
051                              30,  {                        tRightParen     }
052                             615,
053                            4010,  {                                        }
054         {    950   }        4,  {                        Input           }
055                              98,  {                        tReturns        }
056                             615,
057                            3554,  {                                        }
058         {    954   }        6,  {                        Emit            }
059                             113,  {                        aReturns        }
060         {    956   }        4,  {                        Input           }
061                               0,  {                        tIdent          }
062                             615,
063                            3570,  {                                        }
064         {    960   }        6,  {                        Emit            }
065                               0,  {                        aIdent          }
066         {    962   }        9,  {                        oEmitIdent      }
067         {    963   }        1,  {                        Return          }
068         {    964   }        6,  { TypeDeclaration        Emit            }
069                             118,  {                        aType           }
070         {    966   }        4,  {                        Input           }
071                               0,  {                        tIdent          }
072                             621,
073                            4018,  {                                        }
074         {    970   }        6,  {                        Emit            }
075                               0,  {                        aIdent          }
076         {    972   }        9,  {                        oEmitIdent      }
077         {    973   }        5,  {                        InputChoice     }
078                            1008,  {                        TABLE           }
079                             622,
080                            4018,  {                                        }
081         {    977   }        6,  {                        Emit            }
082                             103,  {                        aParms          }
083         {    979   }        0,  {                        Call            }
084                            1067,  {                        TypeFormal      }
085                            4018,  {                                        }
086         {    982   }        5,  {                        InputChoice     }
087                             995,  {                        TABLE           }
088                             625,
089                            4018,  {                                        }
090         {    986   }        3,  {                        Merge           }
091                            1004,  {                                        }
092         {    988   }        3,  {                        Merge           }
```

```
093                          1002,  {                          }
094     {   990  }             0,  {  Call                      }
095                          1067,  {  TypeFormal                }
096                          4018,  {                          }
097     {   993  }             3,  {  Merge                     }
098                          1002,  {                          }
099     {   995  }             2,  {  TABLE BEGINS              }
100                            30,  {  tRightParen              }
101                           986,
102                            34,  {  tComma                   }
103                           990,
104     {  1000  }             2,  {  Repeat                    }
105                           986,  {                          }
106     {  1002  }             2,  {  Repeat                    }
107                           982,  {                          }
108     {  1004  }             6,  {  Emit                      }
109                            53,  {  aEndParms                }
110     {  1006  }             3,  {  Merge                     }
111                          1011,  {                          }
112     {  1008  }             1,  {  TABLE BEGINS              }
113                            29,  {  tLeftParen               }
114                           977,
115     {  1011  }             4,  {  Input                     }
116                            23,  {  tEqual                   }
117                           634,
118                          4034,  {                          }
119     {  1015  }             4,  {  Input                     }
120                            18,  {  tNewLine                 }
121                           634,
122                          4042,  {                          }
123     {  1019  }             5,  {  InputChoice               }
124                          1048,  {  TABLE                   }
125                           635,
126                          4042,  {                          }
127     {  1023  }             5,  {  InputChoice               }
128                          1039,  {  TABLE                   }
129                           637,
130                          4050,  {                          }
131     {  1027  }            14,  {  oEmitLine                 }
132     {  1028  }             6,  {  Emit                      }
133                           107,  {  aPre                     }
134     {  1030  }             0,  {  Call                      }
135                          3000,  {  Expression              }
136                          4058,  {                          }
137     {  1033  }             4,  {  Input                     }
138                            30,  {  tRightParen              }
139                           639,
140                          4066,  {                          }
141     {  1037  }             3,  {  Merge                     }
142                          1042,  {                          }
143     {  1039  }             1,  {  TABLE BEGINS              }
144                            29,  {  tLeftParen               }
```

```
145                          1027,
146     {  1042  }              4,  {                  Input          }
147                             35,  {                  tSemicolon     }
148                            642,
149                           4074,  {                                 }
150     {  1046  }              3,  {                  Merge          }
151                           1051,  {                                 }
152     {  1048  }              1,  {                  TABLE BEGINS   }
153                             93,  {                  tPre           }
154                           1023,
155     {  1051  }             14,  {                  oEmitLine      }
156     {  1052  }              5,  {                  InputChoice    }
157                           1060,  {                  TABLE          }
158                            646,
159                           4074,  {                                 }
160     {  1056  }              6,  {                  Emit           }
161                             66,  {                  aForward       }
162     {  1058  }              3,  {                  Merge          }
163                           1066,  {                                 }
164     {  1060  }              1,  {                  TABLE BEGINS   }
165                             71,  {                  tForward       }
166                           1056,
167     {  1063  }              0,  {                  Call           }
168                           1140,  {                  TypeDefinition }
169                           3570,  {                                 }
170     {  1066  }              1,  {                  Return         }
171     {  1067  }              0,  { TypeFormal       Call           }
172                            287,  {                  Pervasive      }
173                           4082,  {                                 }
174     {  1070  }              5,  {                  InputChoice    }
175                           1076,  {                  TABLE          }
176                            658,
177                           4082,  {                                 }
178     {  1074  }              3,  {                  Merge          }
179                           1079,  {                                 }
180     {  1076  }              1,  {                  TABLE BEGINS   }
181                             56,  {                  tConst         }
182                           1074,
183     {  1079  }              4,  {                  Input          }
184                              0,  {                  tIdent         }
185                            662,
186                           4090,  {                                 }
187     {  1083  }              6,  {                  Emit           }
188                              0,  {                  aIdent         }
189     {  1085  }              9,  {                  oEmitIdent     }
190     {  1086  }              5,  {                  InputChoice    }
191                           1099,  {                  TABLE          }
192                            663,
193                           4090,  {                                 }
194     {  1090  }              4,  {                  Input          }
195                              0,  {                  tIdent         }
196                            665,
```

```
197                          4090,  {                                  }
198        {  1094  }           6,  {                    Emit         }
199                             0,  {                    aIdent       }
200        {  1096  }           9,  {                    oEmitIdent   }
201        {  1097  }           3,  {                    Merge        }
202                          1104,  {                                  }
203        {  1099  }           1,  {                    TABLE BEGINS }
204                            34,  {                    tComma       }
205                          1090,
206        {  1102  }           3,  {                    Merge        }
207                          1106,  {                                  }
208        {  1104  }           2,  {                    Repeat       }
209                          1086,  {                                  }
210        {  1106  }           4,  {                    Input        }
211                            36,  {                    tColon       }
212                           669,
213                          4098,  {                                  }
214        {  1110  }           6,  {                    Emit         }
215                           102,  {                    aParmType    }
216        {  1112  }           0,  {                    Call         }
217                          1418,  {                    IndexType    }
218                          3570,  {                                  }
219        {  1115  }           1,  {                    Return       }
220        {  1116  }           6,  { InlineProcedure    Emit         }
221                            80,  {                    aInline      }
222        {  1118  }           5,  {                    InputChoice  }
223                          1132,  {                    TABLE        }
224                           676,
225                          4106,  {                                  }
226        {  1122  }           0,  {                    Call         }
227                          2006,  {                    ProcedureDeclar }
228                          3570,  {                                  }
229        {  1125  }           3,  {                    Merge        }
230                          1139,  {                                  }
231        {  1127  }           0,  {                    Call         }
232                          2030,  {                    FunctionDeclara }
233                          3570,  {                                  }
234        {  1130  }           3,  {                    Merge        }
235                          1139,  {                                  }
236        {  1132  }           2,  {                    TABLE BEGINS }
237                            94,  {                    tProcedure   }
238                          1122,
239                            73,  {                    tFunction    }
240                          1127,
241        {  1137  }           2,  {                    Repeat       }
242                          1122,  {                                  }
243        {  1139  }           1,  {                    Return       }
244        {  1140  }           5,  { TypeDefinition     InputChoice  }
245                          1391,  {                    TABLE        }
246                           687,
247                          4114,  {                                  }
248        {  1144  }           0,  {                    Call         }
```

```
249                          1434,  {                EnumeratedTypeD }
250                          3570,  {                                }
251      {  1147  }             3,  {                Merge           }
252                          1417,  {                                }
253      {  1149  }             6,  {                Emit            }
254                            17,  {                aArray          }
255      {  1151  }             0,  {                Call            }
256                          1418,  {                IndexType       }
257                          4122,  {                                }
258      {  1154  }             4,  {                Input           }
259                            86,  {                tOf             }
260                           691,
261                          3874,  {                                }
262      {  1158  }             0,  {                Call            }
263                          1140,  {                TypeDefinition  }
264                          3570,  {                                }
265      {  1161  }             3,  {                Merge           }
266                          1417,  {                                }
267      {  1163  }             6,  {                Emit            }
268                           110,  {                aRecord         }
269      {  1165  }             0,  {                Call            }
270                          1597,  {                RecordBody      }
271                          4138,  {                                }
272      {  1168  }             0,  {                Call            }
273                          1798,  {                RecordTrailer   }
274                          3570,  {                                }
275      {  1171  }             3,  {                Merge           }
276                          1417,  {                                }
277      {  1173  }             6,  {                Emit            }
278                            90,  {                aModule         }
279      {  1175  }             0,  {                Call            }
280                            18,  {                ModuleBody      }
281                          4138,  {                                }
282      {  1178  }             0,  {                Call            }
283                          1825,  {                ModuleTrailer   }
284                          3570,  {                                }
285      {  1181  }             3,  {                Merge           }
286                          1417,  {                                }
287      {  1183  }             6,  {                Emit            }
288                           114,  {                aSet            }
289      {  1185  }             4,  {                Input           }
290                            86,  {                tOf             }
291                           697,
292                          4098,  {                                }
293      {  1189  }             0,  {                Call            }
294                          1418,  {                IndexType       }
295                          3570,  {                                }
296      {  1192  }             3,  {                Merge           }
297                          1417,  {                                }
298      {  1194  }             4,  {                Input           }
299                            61,  {                tDependent      }
300                           699,
```

```
301                          4146,  {                              }
302     {  1198  }              5,  {  InputChoice                   }
303                          1240,  {  TABLE                       }
304                           700,
305                          4146,  {                              }
306     {  1202  }              6,  {  Emit                        }
307                             87,  {  aMDRecord                  }
308     {  1204  }              5,  {  InputChoice                 }
309                          1219,  {  TABLE                       }
310                           703,
311                          4154,  {                              }
312     {  1208  }              4,  {  Input                       }
313                             83,  {  tMod                       }
314                           705,
315                          4162,  {                              }
316     {  1212  }              6,  {  Emit                        }
317                             13,  {  aAligned                   }
318     {  1214  }              0,  {  Call                        }
319                          3000,  {  Expression                  }
320                          4170,  {                              }
321     {  1217  }              3,  {  Merge                       }
322                          1222,  {                              }
323     {  1219  }              1,  {  TABLE BEGINS                }
324                             40,  {  tAligned                   }
325                          1208,
326     {  1222  }              0,  {  Call                        }
327                          1852,  {  MDRecordBody                }
328                          4138,  {                              }
329     {  1225  }              0,  {  Call                        }
330                          1798,  {  RecordTrailer               }
331                          3570,  {                              }
332     {  1228  }              3,  {  Merge                       }
333                          1247,  {                              }
334     {  1230  }              6,  {  Emit                        }
335                             86,  {  aMDModule                  }
336     {  1232  }              0,  {  Call                        }
337                             18,  {  ModuleBody                 }
338                          4138,  {                              }
339     {  1235  }              0,  {  Call                        }
340                          1825,  {  ModuleTrailer               }
341                          3570,  {                              }
342     {  1238  }              3,  {  Merge                       }
343                          1247,  {                              }
344     {  1240  }              2,  {  TABLE BEGINS                }
345                             96,  {  tRecord                    }
346                          1202,
347                             84,  {  tModule                    }
348                          1230,
349     {  1245  }              2,  {  Repeat                      }
350                          1202,  {                              }
351     {  1247  }              3,  {  Merge                       }
352                          1417,  {                              }
```

```
353         {   1249   }        6,  {                   Emit                }
354                             29,  {                   aCollection         }
355         {   1251   }         0,  {                   Call                }
356                           1978,  {                   CollectionTypeD     }
357                           3570,  {                                       }
358         {   1254   }         3,  {                   Merge               }
359                           1417,  {                                       }
360         {   1256   }         6,  {                   Emit                }
361                             29,  {                   aCollection         }
362         {   1258   }         6,  {                   Emit                }
363                             34,  {                   aCounted            }
364         {   1260   }         5,  {                   InputChoice         }
365                           1266,  {                   TABLE               }
366                            716,
367                           4194,  {                                       }
368         {   1264   }         3,  {                   Merge               }
369                           1278,  {                                       }
370         {   1266   }         1,  {                   TABLE BEGINS        }
371                             55,  {                   tCollection         }
372                           1264,
373         {   1269   }         6,  {                   Emit                }
374                             33,  {                   aCountMax           }
375         {   1271   }         0,  {                   Call                }
376                           3000,  {                   Expression          }
377                           4202,  {                                       }
378         {   1274   }         4,  {                   Input               }
379                             55,  {                   tCollection         }
380                            719,
381                           4186,  {                                       }
382         {   1278   }         0,  {                   Call                }
383                           1978,  {                   CollectionTypeD     }
384                           3570,  {                                       }
385         {   1281   }         3,  {                   Merge               }
386                           1417,  {                                       }
387         {   1283   }         4,  {                   Input               }
388                             55,  {                   tCollection         }
389                            723,
390                           4186,  {                                       }
391         {   1287   }         6,  {                   Emit                }
392                             29,  {                   aCollection         }
393         {   1289   }         6,  {                   Emit                }
394                             26,  {                   aCheckable          }
395         {   1291   }         0,  {                   Call                }
396                           1978,  {                   CollectionTypeD     }
397                           3570,  {                                       }
398         {   1294   }         3,  {                   Merge               }
399                           1417,  {                                       }
400         {   1296   }         6,  {                   Emit                }
401                            105,  {                   aPointer            }
402         {   1298   }         4,  {                   Input               }
403                              0,  {                   tIdent              }
404                            726,
```

```
                          3938,  {                         }
      {  1302  }             0,  {       Call              }
                          3407,  {       Variable          }
                          3570,  {                         }
      {  1305  }             3,  {       Merge             }
                          1417,  {                         }
      {  1307  }             6,  {       Emit              }
                            99,  {       aPacked           }
      {  1309  }             5,  {       InputChoice       }
                          1376,  {       TABLE             }
                           729,
                          4210,  {                         }
      {  1313  }             6,  {       Emit              }
                            17,  {       aArray            }
      {  1315  }             0,  {       Call              }
                          1418,  {       IndexType         }
                          4122,  {                         }
      {  1318  }             4,  {       Input             }
                            86,  {       tOf               }
                           731,
                          3874,  {                         }
      {  1322  }             0,  {       Call              }
                          1140,  {       TypeDefinition    }
                          3570,  {                         }
      {  1325  }             3,  {       Merge             }
                          1389,  {                         }
      {  1327  }             6,  {       Emit              }
                           110,  {       aRecord           }
      {  1329  }             0,  {       Call              }
                          1597,  {       RecordBody        }
                          4138,  {                         }
      {  1332  }             0,  {       Call              }
                          1798,  {       RecordTrailer     }
                          3570,  {                         }
      {  1335  }             3,  {       Merge             }
                          1389,  {                         }
      {  1337  }             6,  {       Emit              }
                            90,  {       aModule           }
      {  1339  }             0,  {       Call              }
                            18,  {       ModuleBody        }
                          4138,  {                         }
      {  1342  }             0,  {       Call              }
                          1825,  {       ModuleTrailer     }
                          3570,  {                         }
      {  1345  }             3,  {       Merge             }
                          1389,  {                         }
      {  1347  }             6,  {       Emit              }
                           114,  {       aSet              }
      {  1349  }             4,  {       Input             }
                            86,  {       tOf               }
                           737,
                          4098,  {                         }
```

```
457     {  1353  }      0,  {   Call            }
458                  1418,  {   IndexType       }
459                  3570,  {                   }
460     {  1356  }      3,  {   Merge           }
461                  1389,  {                   }
462     {  1358  }      4,  {   Input           }
463                    61,  {   tDependent      }
464                   739,
465                  4218,  {                   }
466     {  1362  }      4,  {   Input           }
467                    84,  {   tModule         }
468                   739,
469                  4226,  {                   }
470     {  1366  }      6,  {   Emit            }
471                    86,  {   aModule         }
472     {  1368  }      0,  {   Call            }
473                    18,  {   ModuleBody      }
474                  4138,  {                   }
475     {  1371  }      0,  {   Call            }
476                  1825,  {   ModuleTrailer   }
477                  3570,  {                   }
478     {  1374  }      3,  {   Merge           }
479                  1389,  {                   }
480     {  1376  }      5,  {   TABLE BEGINS    }
481                    44,  {   tArray          }
482                  1313,
483                    96,  {   tRecord         }
484                  1327,
485                    84,  {   tModule         }
486                  1337,
487                    99,  {   tSet            }
488                  1347,
489                    82,  {   tMachine        }
490                  1358,
491     {  1387  }      2,  {   Repeat          }
492                  1313,  {                   }
493     {  1389  }      3,  {   Merge           }
494                  1417,  {                   }
495     {  1391  }     11,  {   TABLE BEGINS    }
496                    29,  {   tLeftParen      }
497                  1144,
498                    44,  {   tArray          }
499                  1149,
500                    96,  {   tRecord         }
501                  1163,
502                    84,  {   tModule         }
503                  1173,
504                    99,  {   tSet            }
505                  1183,
506                    82,  {   tMachine        }
507                  1194,
508                    55,  {   tCollection     }
```

```
509                                1249,
510                                  58,  {                        tCounted          }
511                                1256,
512                                  52,  {                        tCheckable        }
513                                1283,
514                                  37,  {                        tUpArrow          }
515                                1296,
516                                  89,  {                        tPacked           }
517                                1307,
518       {   1414   }               0,  {                        Call              }
519                                1472,  {                        NamedOrRangeTyp   }
520                                3570,  {                                          }
521       {   1417   }               1,  {                        Return            }
522       {   1418   }               5,  { IndexType              InputChoice       }
523                                1427,  {                        TABLE             }
524                                 749,
525                                4098,  {                                          }
526       {   1422   }               0,  {                        Call              }
527                                1434,  {                        EnumeratedTypeD   }
528                                3570,  {                                          }
529       {   1425   }               3,  {                        Merge             }
530                                1433,  {                                          }
531       {   1427   }               1,  {                        TABLE BEGINS      }
532                                  29,  {                        tLeftParen        }
533                                1422,
534       {   1430   }               0,  {                        Call              }
535                                1472,  {                        NamedOrRangeTyp   }
536                                3570,  {                                          }
537       {   1433   }               1,  {                        Return            }
538       {   1434   }               6,  { EnumeratedTypeD        Emit              }
539                                  58,  {                        aEnum             }
540       {   1436   }               4,  {                        Input             }
541                                  0,  {                        tIdent            }
542                                 760,
543                                4234,  {                                          }
544       {   1440   }               6,  {                        Emit              }
545                                  0,  {                        aIdent            }
546       {   1442   }               9,  {                        oEmitIdent        }
547       {   1443   }               5,  {                        InputChoice       }
548                                1460,  {                        TABLE             }
549                                 761,
550                                4234,  {                                          }
551       {   1447   }               3,  {                        Merge             }
552                                1469,  {                                          }
553       {   1449   }               3,  {                        Merge             }
554                                1467,  {                                          }
555       {   1451   }               4,  {                        Input             }
556                                  0,  {                        tIdent            }
557                                 765,
558                                4234,  {                                          }
559       {   1455   }               6,  {                        Emit              }
560                                  0,  {                        aIdent            }
```

```
561     {  1457  }     9,  {                    oEmitIdent      }
562     {  1458  }     3,  {                    Merge           }
563                 1467,  {                                    }
564     {  1460  }     2,  {                    TABLE BEGINS    }
565                   30,  {                    tRigntParen     }
566                 1447,
567                   34,  {                    tComma          }
568                 1451,
569     {  1465  }     2,  {                    Repeat          }
570                 1447,  {                                    }
571     {  1467  }     2,  {                    Repeat          }
572                 1443,  {                                    }
573     {  1469  }     6,  {                    Emit            }
574                   43,  {                    aEndEnum        }
575     {  1471  }     1,  {                    Return          }
576     {  1472  }     5,  { NamedOrRangeTyp    InputChoice     }
577                 1581,  {                    TABLE           }
578                  773,
579                 4242,  {                                    }
580     {  1476  }     0,  {                    Call            }
581                 3407,  {                    Variable        }
582                 4098,  {                                    }
583     {  1479  }     5,  {                    InputChoice     }
584                 1487,  {                    TABLE           }
585                  776,
586                 4098,  {                                    }
587     {  1483  }     6,  {                    Emit            }
588                   45,  {                    aEndExpression  }
589     {  1485  }     3,  {                    Merge           }
590                 1574,  {                                    }
591     {  1487  }     1,  {                    TABLE BEGINS    }
592                   32,  {                    tDoublePeriod   }
593                 1483,
594     {  1490  }     5,  {                    InputChoice     }
595                 1554,  {                    TABLE           }
596                  780,
597                 4098,  {                                    }
598     {  1494  }     0,  {                    Call            }
599                 3283,  {                    Term            }
600                 4098,  {                                    }
601     {  1497  }     6,  {                    Emit            }
602                   12,  {                    aAdd            }
603     {  1499  }     0,  {                    Call            }
604                 3237,  {                    PartialSum      }
605                 4098,  {                                    }
606     {  1502  }     3,  {                    Merge           }
607                 1570,  {                                    }
608     {  1504  }     0,  {                    Call            }
609                 3283,  {                    Term            }
610                 4098,  {                                    }
611     {  1507  }     6,  {                    Emit            }
612                  116,  {                    aSubtract       }
```

```
613         {   1509   }        0,  {                Call            }
614                             3237,  {                PartialSum      }
615                             4098,  {                                }
616         {   1512   }        3,  {                Merge           }
617                             1570,  {                                }
618         {   1514   }        0,  {                Call            }
619                             3283,  {                Term            }
620                             4098,  {                                }
621         {   1517   }        6,  {                Emit            }
622                              126,  {                aXor            }
623         {   1519   }        0,  {                Call            }
624                             3237,  {                PartialSum      }
625                             4098,  {                                }
626         {   1522   }        3,  {                Merge           }
627                             1570,  {                                }
628         {   1524   }        0,  {                Call            }
629                             3327,  {                Factor          }
630                             4098,  {                                }
631         {   1527   }        6,  {                Emit            }
632                               92,  {                aMultiply       }
633         {   1529   }        0,  {                Call            }
634                             3237,  {                PartialSum      }
635                             4098,  {                                }
636         {   1532   }        3,  {                Merge           }
637                             1570,  {                                }
638         {   1534   }        0,  {                Call            }
639                             3327,  {                Factor          }
640                             4098,  {                                }
641         {   1537   }        6,  {                Emit            }
642                               37,  {                aDiv            }
643         {   1539   }        0,  {                Call            }
644                             3237,  {                PartialSum      }
645                             4098,  {                                }
646         {   1542   }        3,  {                Merge           }
647                             1570,  {                                }
648         {   1544   }        0,  {                Call            }
649                             3327,  {                Factor          }
650                             4098,  {                                }
651         {   154/   }        6,  {                Emit            }
652                               89,  {                aMod            }
653         {   1549   }        0,  {                Call            }
654                             3237,  {                PartialSum      }
655                             4098,  {                                }
656         {   1552   }        3,  {                Merge           }
657                             1570,  {                                }
658         {   1554   }        6,  {                TABLE BEGINS    }
659                               20,  {                tPlus           }
660                             1494,
661                               21,  {                tMinus          }
662                             1504,
663                              108,  {                tXor            }
664                             1514,
```

```
665                          22,  {                    tAsterisk          }
666                        1524,
667                          62,  {                    tDiv               }
668                        1534,
669                          83,  {                    tMod               }
670                        1544,
671     {  1567  }            6,  {                    Emit               }
672                          119, {                    aTypeName          }
673     {  1569  }            1,  {                    Return             }
674     {  1570  }            4,  {                    Input              }
675                          32,  {                    tDoublePeriod      }
676                         796,
677                        4274,  {                                       }
678     {  1574  }            6,  {                    Emit               }
679                          117, {                    aTo                }
680     {  1576  }            0,  {                    Call               }
681                        3224,  {                    Sum                }
682                        3570,  {                                       }
683     {  1579  }            3,  {                    Merge              }
684                        1596,  {                                       }
685     {  1581  }            1,  {                    TABLE BEGINS       }
686                           0,  {                    tIdent             }
687                        1476,
688     {  1584  }            0,  {                    Call               }
689                        3224,  {                    Sum                }
690                        4098,  {                                       }
691     {  1587  }            4,  {                    Input              }
692                          32,  {                    tDoublePeriod      }
693                         800,
694                        4274,  {                                       }
695     {  1591  }            6,  {                    Emit               }
696                          117, {                    aTo                }
697     {  1593  }            0,  {                    Call               }
698                        3224,  {                    Sum                }
699                        3570,  {                                       }
700     {  1596  }            1,  {                    Return             }
701     {  1597  }            0,  { RecordBody         Call               }
702                        1741,  {                    DeclarationsInR    }
703                        4282,  {                                       }
704     {  1600  }            5,  {                    InputChoice        }
705                        1737,  {                    TABLE              }
706                         808,
707                        4282,  {                                       }
708     {  1604  }            6,  {                    Emit               }
709                          25,  {                    aCase              }
710     {  1606  }            4,  {                    Input              }
711                           0,  {                    tIdent             }
712                         810,
713                        4282,  {                                       }
714     {  1610  }            6,  {                    Emit               }
715                           0,  {                    aIdent             }
716     {  1612  }            9,  {                    oEmitIdent         }
```

```
717     {  1613  }      5,  {                   InputChoice     }
718                    1624,  {                 TABLE           }
719                     811,
720                    4282,  {                                 }
721     {  1617  }      6,  {                   Emit            }
722                      36,  {                 aDefault        }
723     {  1619  }       0,  {                 Call            }
724                    3000,  {                 Expression      }
725                    4290,  {                                 }
726     {  1622  }       3,  {                 Merge           }
727                    1627,  {                                 }
728     {  1624  }       1,  {                 TABLE BEGINS    }
729                      60,  {                 tDefault        }
730                    1617,
731     {  1627  }       4,  {                 Input           }
732                      86,  {                 tOf             }
733                     816,
734                    4298,  {                                 }
735     {  1631  }       4,  {                 Input           }
736                      18,  {                 tNewLine        }
737                     816,
738                    4298,  {                                 }
739     {  1635  }      14,  {                 oEmitLine       }
740     {  1636  }       5,  {                 InputChoice     }
741                    1673,  {                 TABLE           }
742                     818,
743                    4298,  {                                 }
744     {  1640  }       4,  {                 Input           }
745                      51,  {                 tCase           }
746                     820,
747                    3562,  {                                 }
748     {  1644  }       3,  {                 Merge           }
749                    1728,  {                                 }
750     {  1646  }       3,  {                 Merge           }
751                    1726,  {                                 }
752     {  1648  }       4,  {                 Input           }
753                      38,  {                 tLabel          }
754                     822,
755                    4306,  {                                 }
756     {  1652  }       4,  {                 Input           }
757                      18,  {                 tNewLine        }
758                     822,
759                    4314,  {                                 }
760     {  1656  }       6,  {                 Emit            }
761                      98,  {                 aOtherwise      }
762     {  1658  }       0,  {                 Call            }
763                    1741,  {                 DeclarationsInR }
764                    4322,  {                                 }
765     {  1661  }       4,  {                 Input           }
766                      65,  {                 tEnd            }
767                     824,
768                    4330,  {                                 }
```

```
769     {  1665  }      4,  {                   Input           }
770                    51,  {                   tCase           }
771                   824,
772                  3562,  {                                   }
773     {  1669  }      3,  {                   Merge           }
774                  1728,  {                                   }
775     {  1671  }      3,  {                   Merge           }
776                  1726,  {                                   }
777     {  1673  }      2,  {                   TABLE BEGINS    }
778                    65,  {                   tEnd            }
779                  1640,
780                    88,  {                   tOtherwise      }
781                  1648,
782     {  1678  }      6,  {                   Emit            }
783                    82,  {                   aLabels         }
784     {  1680  }      0,  {                   Call            }
785                  2982,  {                   CaseLabel       }
786                  4298,  {                                   }
787     {  1683  }      5,  {                   InputChoice     }
788                  1692,  {                   TABLE           }
789                   827,
790                  4298,  {                                   }
791     {  1687  }      0,  {                   Call            }
792                  2982,  {                   CaseLabel       }
793                  4298,  {                                   }
794     {  1690  }      3,  {                   Merge           }
795                  1697,  {                                   }
796     {  1692  }      1,  {                   TABLE BEGINS    }
797                    34,  {                   tComma          }
798                  1687,
799     {  1695  }      3,  {                   Merge           }
800                  1699,  {                                   }
801     {  1697  }      2,  {                   Repeat          }
802                  1683,  {                                   }
803     {  1699  }      6,  {                   Emit            }
804                    50,  {                   aEndLabels      }
805     {  1701  }      4,  {                   Input           }
806                    38,  {                   tLabel          }
807                   834,
808                  4298,  {                                   }
809     {  1705  }      4,  {                   Input           }
810                    18,  {                   tNewLine        }
811                   834,
812                  4298,  {                                   }
813     {  1709  }      0,  {                   Call            }
814                  1741,  {                   DeclarationsInR }
815                  4298,  {                                   }
816     {  1712  }      4,  {                   Input           }
817                    65,  {                   tEnd            }
818                   836,
819                  4298,  {                                   }
820     {  1716  }      6,  {                   Emit            }
```

821		49,	{	aEndLabel }
822	{ 1718 }	0,	{	Call }
823		2982,	{	CaseLabel }
824		4298,	{	}
825	{ 1721 }	4,	{	Input }
826		35,	{	tSemicolon }
827		836,		
828		4298,	{	}
829	{ 1725 }	14,	{	oEmitLine }
830	{ 1726 }	2,	{	Repeat }
831		1636,	{	}
832	{ 1728 }	6,	{	Emit }
833		41,	{	aEndCase }
834	{ 1730 }	4,	{	Input }
835		35,	{	tSemicolon }
836		839,		
837		3570,	{	}
838	{ 1734 }	14,	{	oEmitLine }
839	{ 1735 }	3,	{	Merge }
840		1740,	{	}
841	{ 1737 }	1,	{	TABLE BEGINS }
842		51,	{	tCase }
843		1604,		
844	{ 1740 }	1,	{	Return }
845	{ 1741 }	14,	{ DeclarationsInR	oEmitLine }
846	{ 1742 }	5,	{	InputChoice }
847		1774,	{	TABLE }
848		850,		
849		4346,	{	}
850	{ 1746 }	0,	{	Call }
851		629,	{	VariableDeclara }
852		4346,	{	}
853	{ 1749 }	3,	{	Merge }
854		1791,	{	}
855	{ 1751 }	0,	{	Call }
856		789,	{	ConstantDeclara }
857		4346,	{	}
858	{ 1754 }	3,	{	Merge }
859		1791,	{	}
860	{ 1756 }	6,	{	Emit }
861		104,	{	aPervasive }
862	{ 1758 }	4,	{	Input }
863		56,	{	tConst }
864		856,		
865		4346,	{	}
866	{ 1762 }	0,	{	Call }
867		789,	{	ConstantDeclara }
868		4346,	{	}
869	{ 1765 }	3,	{	Merge }
870		1791,	{	}
871	{ 1767 }	0,	{	Call }
872		2503,	{	AssertStatement }

```
873                          4346,  {                                 }
874     {  1770  }              3,  {  Merge                          }
875                          1791,  {                                 }
876     {  1772  }              3,  {  Merge                          }
877                          1791,  {                                 }
878     {  1774  }              5,  {  TABLE BEGINS                   }
879                           105,  {  tVar                           }
880                          1746,
881                            56,  {  tConst                         }
882                          1751,
883                            91,  {  tPervasive                     }
884                          1756,
885                            45,  {  tAssert                        }
886                          1767,
887                            35,  {  tSemicolon                     }
888                          1772,
889     {  1785  }              4,  {  Input                          }
890                            35,  {  tSemicolon                     }
891                           861,
892                          3562,  {                                 }
893     {  1789  }              3,  {  Merge                          }
894                          1797,  {                                 }
895     {  1791  }              4,  {  Input                          }
896                            35,  {  tSemicolon                     }
897                           863,
898                          3570,  {                                 }
899     {  1795  }              2,  {  Repeat                         }
900                          1741,  {                                 }
901     {  1797  }              1,  {  Return                         }
902     {  1798  }              4,  {  RecordTrailer  Input           }
903                            65,  {  tEnd                           }
904                           870,
905                          4354,  {                                 }
906     {  1802  }              6,  {  Emit                           }
907                            54,  {  aEndRecord                     }
908     {  1804  }              5,  {  InputChoice                    }
909                          1817,  {  TABLE                         }
910                           871,
911                          4354,  {                                 }
912     {  1808  }              3,  {  Merge                          }
913                          1824,  {                                 }
914     {  1810  }              6,  {  Emit                           }
915                            46,  {  aEndIdent                      }
916     {  1812  }              6,  {  Emit                           }
917                             0,  {  aIdent                         }
918     {  1814  }              9,  {  oEmitIdent                     }
919     {  1815  }              3,  {  Merge                          }
920                          1824,  {                                 }
921     {  1817  }              2,  {  TABLE BEGINS                   }
922                            96,  {  tRecord                        }
923                          1808,
924                             0,  {  tIdent                         }
```

```
925                          1810,
926     {  1822  }              2,  {                       Repeat          }
927                          1808,  {                                       }
928     {  1824  }              1,  {                       Return          }
929     {  1825  }              4,  { ModuleTrailer         Input           }
930                             65,  {                      tEnd            }
931                            881,
932                           4362,  {                                       }
933     {  1829  }              6,  {                       Emit            }
934                             52,  {                      aEndModule      }
935     {  1831  }              5,  {                       InputChoice     }
936                           1844,  {                      TABLE           }
937                            882,
938                           4362,  {                                       }
939     {  1835  }              3,  {                       Merge           }
940                           1851,  {                                       }
941     {  1837  }              6,  {                       Emit            }
942                             46,  {                      aEndIdent       }
943     {  1839  }              6,  {                       Emit            }
944                              0,  {                      aIdent          }
945     {  1841  }              9,  {                       oEmitIdent      }
946     {  1842  }              3,  {                       Merge           }
947                           1851,  {                                       }
948     {  1844  }              2,  {                       TABLE BEGINS    }
949                             84,  {                      tModule         }
950                           1835,
951                              0,  {                      tIdent          }
952                           1837,
953     {  1849  }              2,  {                       Repeat          }
954                           1835,  {                                       }
955     {  1851  }              1,  {                       Return          }
956     {  1852  }             14,  { MDRecordBody          oEmitLine       }
957     {  1853  }              5,  {                       InputChoice     }
958                           1954,  {                      TABLE           }
959                            894,
960                           4370,  {                                       }
961     {  1857  }              6,  {                       Emit            }
962                            122,  {                      aVar            }
963     {  1859  }              4,  {                       Input           }
964                              0,  {                      tIdent          }
965                            896,
966                           4370,  {                                       }
967     {  1863  }              6,  {                       Emit            }
968                              0,  {                      aIdent          }
969     {  1865  }              9,  {                       oEmitIdent      }
970     {  1866  }              4,  {                       Input           }
971                             29,  {                      tLeftParen      }
972                            897,
973                           4370,  {                                       }
974     {  1870  }              4,  {                       Input           }
975                             46,  {                      tAt             }
976                            897,
```

```
977                          4370,  {                          }
978     {  1874  }              6,  {  Emit                    }
979                            20,  {  aAt                      }
980     {  1876  }              0,  {  Call                    }
981                          3000,  {  Expression              }
982                          4370,  {                          }
983     {  1879  }              5,  {  InputChoice             }
984                          1899,  {  TABLE                   }
985                           898,
986                          4370,  {                          }
987     {  1883  }              6,  {  Emit                    }
988                            23,  {  aBits                    }
989     {  1885  }              0,  {  Call                    }
990                          3224,  {  Sum                     }
991                          4370,  {                          }
992     {  1888  }              4,  {  Input                   }
993                            32,  {  tDoublePeriod           }
994                           900,
995                          4370,  {                          }
996     {  1892  }              6,  {  Emit                    }
997                           117,  {  aTo                      }
998     {  1894  }              0,  {  Call                    }
999                          3224,  {  Sum                     }
000                          4370,  {                          }
001     {  1897  }              3,  {  Merge                   }
002                          1902,  {                          }
003     {  1899  }              1,  {  TABLE BEGINS            }
004                            49,  {  tBits                    }
005                          1883,
006     {  1902  }              4,  {  Input                   }
007                            30,  {  tRightParen             }
008                           903,
009                          4370,  {                          }
010     {  1906  }              4,  {  Input                   }
011                            36,  {  tColon                   }
012                           903,
013                          4370,  {                          }
014     {  1910  }              6,  {  Emit                    }
015                           123,  {  aVarType                 }
016     {  1912  }              0,  {  Call                    }
017                          1140,  {  TypeDefinition          }
018                          4370,  {                          }
019     {  1915  }              5,  {  InputChoice             }
020                          1926,  {  TABLE                   }
021                           904,
022                          4370,  {                          }
023     {  1919  }              6,  {  Emit                    }
024                            78,  {  aInitial                 }
025     {  1921  }              0,  {  Call                    }
026                          3000,  {  Expression              }
027                          4370,  {                          }
028     {  1924  }              3,  {  Merge                   }
```

```
029                          1929,  {                          }
030        {  1926  }           1,  {  TABLE BEGINS             }
031                            33,  {  tAssign                   }
032                          1919,
033        {  1929  }           3,  {  Merge                    }
034                          1971,  {                          }
035        {  1931  }           0,  {  Call                     }
036                           789,  {  ConstantDeclara         }
037                          4370,  {                          }
038        {  1934  }           3,  {  Merge                    }
039                          1971,  {                          }
040        {  1936  }           6,  {  Emit                     }
041                           104,  {  aPervasive               }
042        {  1938  }           4,  {  Input                    }
043                            56,  {  tConst                    }
044                           912,
045                          4370,  {                          }
046        {  1942  }           0,  {  Call                     }
047                           789,  {  ConstantDeclara         }
048                          4370,  {                          }
049        {  1945  }           3,  {  Merge                    }
050                          1971,  {                          }
051        {  1947  }           0,  {  Call                     }
052                          2503,  {  AssertStatement         }
053                          4370,  {                          }
054        {  1950  }           3,  {  Merge                    }
055                          1971,  {                          }
056        {  1952  }           3,  {  Merge                    }
057                          1971,  {                          }
058        {  1954  }           5,  {  TABLE BEGINS             }
059                           105,  {  tVar                     }
060                          1857,
061                            56,  {  tConst                    }
062                          1931,
063                            91,  {  tPervasive                }
064                          1936,
065                            45,  {  tAssert                   }
066                          1947,
067                            35,  {  tSemicolon                }
068                          1952,
069        {  1965  }           4,  {  Input                    }
070                            35,  {  tSemicolon                }
071                           917,
072                          3562,  {                          }
073        {  1969  }           3,  {  Merge                    }
074                          1977,  {                          }
075        {  1971  }           4,  {  Input                    }
076                            35,  {  tSemicolon                }
077                           919,
078                          3570,  {                          }
079        {  1975  }           2,  {  Repeat                   }
080                          1852,  {                          }
```

```
081         {   1977   }        1,  {                       Return          }
082         {   1978   }        4,  { CollectionTypeD       Input           }
083                             86,  {                       tOf             }
084                            926,
085                           4378,  {                                       }
086         {   1982   }        6,  {                       Emit            }
087                            123,  {                       aVarType        }
088         {   1984   }        0,  {                       Call            }
089                           1140,  {                       TypeDefinition  }
090                           4386,  {                                       }
091         {   1987   }        5,  {                       InputChoice     }
092                           2002,  {                       TABLE           }
093                            927,
094                           4386,  {                                       }
095         {   1991   }        6,  {                       Emit            }
096                             73,  {                       aIn             }
097         {   1993   }        4,  {                       Input           }
098                             0,  {                       tIdent          }
099                            929,
100                           3938,  {                                       }
101         {   1997   }        0,  {                       Call            }
102                           3407,  {                       Variable        }
103                           3570,  {                                       }
104         {   2000   }        3,  {                       Merge           }
105                           2005,  {                                       }
106         {   2002   }        1,  {                       TABLE BEGINS    }
107                             76,  {                       tIn             }
108                           1991,
109         {   2005   }        1,  {                       Return          }
110         {   2006   }        6,  { ProcedureDeclar       Emit            }
111                            108,  {                       aProcedure      }
112         {   2008   }        4,  {                       Input           }
113                             0,  {                       tIdent          }
114                            937,
115                           4394,  {                                       }
116         {   2012   }        6,  {                       Emit            }
117                             0,  {                       aIdent          }
118         {   2014   }        9,  {                       oEmitIdent      }
119         {   2015   }        0,  {                       Call            }
120                           2079,  {                       FormalParameter }
121                           4402,  {                                       }
122         {   2018   }        4,  {                       Input           }
123                            23,  {                       tEqual          }
124                            937,
125                           4410,  {                                       }
126         {   2022   }        4,  {                       Input           }
127                            18,  {                       tNewLine        }
128                            937,
129                           3650,  {                                       }
130         {   2026   }        0,  {                       Call            }
131                           2161,  {                       RoutineDefiniti }
132                           3570,  {                                       }
```

```
133     {  2029  }      1,  {                    Return          }
134     {  2030  }      6,  { FunctionDeclara    Emit            }
135                    67,  {                    aFunction       }
136     {  2032  }      4,  {                    Input           }
137                     0,  {                    tIdent          }
138                   944,
139                  4418,  {                                    }
140     {  2036  }      6,  {                    Emit            }
141                     0,  {                    aIdent          }
142     {  2038  }      9,  {                    oEmitIdent      }
143     {  2039  }      0,  {                    Call            }
144                  2079,  {                    FormalParameter }
145                  4418,  {                                    }
146     {  2042  }      5,  {                    InputChoice     }
147                  2064,  {                    TABLE           }
148                   945,
149                  4418,  {                                    }
150     {  2046  }      6,  {                    Emit            }
151                   113,  {                    aReturns        }
152     {  2048  }      4,  {                    Input           }
153                     0,  {                    tIdent          }
154                   947,
155                  4426,  {                                    }
156     {  2052  }      6,  {                    Emit            }
157                     0,  {                    aIdent          }
158     {  2054  }      9,  {                    oEmitIdent      }
159     {  2055  }      4,  {                    Input           }
160                    36,  {                    tColon          }
161                   947,
162                  4434,  {                                    }
163     {  2059  }      0,  {                    Call            }
164                  1140,  {                    TypeDefinition  }
165                  4402,  {                                    }
166     {  2062  }      3,  {                    Merge           }
167                  2067,  {                                    }
168     {  2064  }      1,  {                    TABLE BEGINS    }
169                    98,  {                    tReturns        }
170                  2046,
171     {  2067  }      4,  {                    Input           }
172                    23,  {                    tEqual          }
173                   950,
174                  4410,  {                                    }
175     {  2071  }      4,  {                    Input           }
176                    18,  {                    tNewLine        }
177                   950,
178                  3650,  {                                    }
179     {  2075  }      0,  {                    Call            }
180                  2161,  {                    RoutineDefiniti }
181                  3570,  {                                    }
182     {  2078  }      1,  {                    Return          }
183     {  2079  }      5,  { FormalParameter    InputChoice     }
184                  2114,  {                    TABLE           }
```

```
                          957,
                         4442,  {                                          }
    {  2083  }              6,  {                          Emit            }
                          103,  {                          aParms          }
    {  2085  }              0,  {                          Call            }
                         2118,  {                          RoutineFormal   }
                         4450,  {                                          }
    {  2088  }              5,  {                          InputChoice     }
                         2103,  {                          TABLE           }
                          960,
                         4450,  {                                          }
    {  2092  }              6,  {                          Emit            }
                           53,  {                          aEndParms       }
    {  2094  }              3,  {                          Merge           }
                         2112,  {                                          }
    {  2096  }              3,  {                          Merge           }
                         2110,  {                                          }
    {  2098  }              0,  {                          Call            }
                         2118,  {                          RoutineFormal   }
                         4450,  {                                          }
    {  2101  }              3,  {                          Merge           }
                         2110,  {                                          }
    {  2103  }              2,  {                          TABLE BEGINS    }
                           30,  {                          tRightParen     }
                         2092,
                           34,  {                          tComma          }
                         2098,
    {  2108  }              2,  {                          Repeat          }
                         2092,  {                                          }
    {  2110  }              2,  {                          Repeat          }
                         2088,  {                                          }
    {  2112  }              3,  {                          Merge           }
                         2117,  {                                          }
    {  2114  }              1,  {                          TABLE BEGINS    }
                           29,  {                          tLeftParen      }
                         2083,
    {  2117  }              1,  {                          Return          }
    {  2118  }              0,  { RoutineFormal            Call            }
                          287,  {                          Pervasive       }
                         4458,  {                                          }
    {  2121  }              0,  {                          Call            }
                          299,  {                          BindingConditio }
                         4466,  {                                          }
    {  2124  }              4,  {                          Input           }
                            0,  {                          tIdent          }
                          973,
                         4466,  {                                          }
    {  2128  }              6,  {                          Emit            }
                            0,  {                          aIdent          }
    {  2130  }              9,  {                          oEmitIdent      }
    {  2131  }              5,  {                          InputChoice     }
                         2144,  {                          TABLE           }
```

```
                                 974,
                                4466,  {                                              }
          {   2135   }             4,  {                        Input                 }
                                   0,  {                        tIdent                }
                                 976,
                                4466,  {                                              }
          {   2139   }             6,  {                        Emit                  }
                                   0,  {                        aIdent                }
          {   2141   }             9,  {                        oEmitIdent            }
          {   2142   }             3,  {                        Merge                 }
                                2149,  {                                              }
          {   2144   }             1,  {                        TABLE BEGINS          }
                                  34,  {                        tComma                }
                                2135,
          {   2147   }             3,  {                        Merge                 }
                                2151,  {                                              }
          {   2149   }             2,  {                        Repeat                }
                                2131,  {                                              }
          {   2151   }             4,  {                        Input                 }
                                  36,  {                        tColon                }
                                 980,
                                3874,  {                                              }
          {   2155   }             6,  {                        Emit                  }
                                 102,  {                        aParmType             }
          {   2157   }             0,  {                        Call                  }
                                1140,  {                        TypeDefinition        }
                                3570,  {                                              }
          {   2160   }             1,  {                        Return                }
          {   2161   }            14,  { RoutineDefiniti        oEmitLine             }
          {   2162   }             0,  {                        Call                  }
                                 183,  {                        ImportClauses         }
                                4474,  {                                              }
          {   2165   }             5,  {                        InputChoice           }
                                2198,  {                        TABLE                 }
                                 988,
                                4474,  {                                              }
          {   2169   }             5,  {                        InputChoice           }
                                2189,  {                        TABLE                 }
                                 990,
                                4482,  {                                              }
          {   2173   }             6,  {                        Emit                  }
                                 107,  {                        aPre                  }
          {   2175   }             0,  {                        Call                  }
                                3000,  {                        Expression            }
                                4482,  {                                              }
          {   2178   }             4,  {                        Input                 }
                                  30,  {                        tRightParen           }
                                 992,
                                4482,  {                                              }
          {   2182   }             4,  {                        Input                 }
                                  35,  {                        tSemicolon            }
                                 992,
```

```
289                          4482,  {                          }
290     {  2186  }             14,  {  oEmitLine                }
291     {  2187  }              3,  {  Merge                    }
292                          2196,  {                          }
293     {  2189  }              1,  {  TABLE BEGINS             }
294                             29,  {  tLeftParen              }
295                          2173,
296     {  2192  }              4,  {  Input                    }
297                             35,  {  tSemicolon              }
298                           995,
299                          4482,  {                          }
300     {  2196  }              3,  {  Merge                    }
301                          2201,  {                          }
302     {  2198  }              1,  {  TABLE BEGINS             }
303                             93,  {  tPre                    }
304                          2169,
305     {  2201  }              5,  {  InputChoice              }
306                          2234,  {  TABLE                    }
307                           999,
308                          4482,  {                          }
309     {  2205  }              5,  {  InputChoice              }
310                          2225,  {  TABLE                    }
311                          1001,
312                          4490,  {                          }
313     {  2209  }              6,  {  Emit                     }
314                           106,  {  aPost                    }
315     {  2211  }              0,  {  Call                     }
316                          3000,  {  Expression               }
317                          4498,  {                          }
318     {  2214  }              4,  {  Input                    }
319                             30,  {  tRightParen             }
320                          1003,
321                          4506,  {                          }
322     {  2218  }              4,  {  Input                    }
323                             35,  {  tSemicolon              }
324                          1003,
325                          4514,  {                          }
326     {  2222  }             14,  {  oEmitLine                }
327     {  2223  }              3,  {  Merge                    }
328                          2232,  {                          }
329     {  2225  }              1,  {  TABLE BEGINS             }
330                             29,  {  tLeftParen              }
331                          2209,
332     {  2228  }              4,  {  Input                    }
333                             35,  {  tSemicolon              }
334                          1006,
335                          4514,  {                          }
336     {  2232  }              3,  {  Merge                    }
337                          2237,  {                          }
338     {  2234  }              1,  {  TABLE BEGINS             }
339                             92,  {  tPost                   }
340                          2205,
```

```
341     {  2237  }      0,  {                    Call            }
342                  2241,  {                    RoutineBody     }
343                  3570,  {                                    }
344     {  2240  }      1,  {                    Return          }
345     {  2241  }      5,  { RoutineBody        InputChoice     }
346                  2335,  {                    TABLE           }
347                  1016,
348                  4522,  {                                    }
349     {  2245  }      6,  {                    Emit            }
350                    21,  {                    aBegin          }
351     {  2247  }      0,  {                    Call            }
352                  2534,  {                    BlockBody       }
353                  4538,  {                                    }
354     {  2250  }      4,  {                    Input           }
355                    65,  {                    tEnd            }
356                  1020,
357                  3554,  {                                    }
358     {  2254  }      6,  {                    Emit            }
359                    39,  {                    aEndBegin       }
360     {  2256  }      5,  {                    InputChoice     }
361                  2267,  {                    TABLE           }
362                  1021,
363                  3554,  {                                    }
364     {  2260  }      6,  {                    Emit            }
365                    46,  {                    aEndIdent       }
366     {  2262  }      6,  {                    Emit            }
367                     0,  {                    aIdent          }
368     {  2264  }      9,  {                    oEmitIdent      }
369     {  2265  }      3,  {                    Merge           }
370                  2270,  {                                    }
371     {  2267  }      1,  {                    TABLE BEGINS    }
372                     0,  {                    tIdent          }
373                  2260,
374     {  2270  }      3,  {                    Merge           }
375                  2346,  {                                    }
376     {  2272  }      6,  {                    Emit            }
377                    28,  {                    aCode           }
378     {  2274  }     14,  {                    oEmitLine       }
379     {  2275  }      5,  {                    InputChoice     }
380                  2302,  {                    TABLE           }
381                  1028,
382                  4546,  {                                    }
383     {  2279  }      5,  {                    InputChoice     }
384                  2288,  {                    TABLE           }
385                  1031,
386                  4554,  {                                    }
387     {  2283  }      6,  {                    Emit            }
388                     6,  {                    aCodeUnit       }
389     {  2285  }     13,  {                    oEmitCode       }
390     {  2286  }      3,  {                    Merge           }
391                  2297,  {                                    }
392     {  2288  }      1,  {                    TABLE BEGINS    }
```

```
393                              15,  {  tCodeMiddle     }
394                            2283,
395     {  2291  }               4,  {  Input           }
396                              16,  {  tCodeEnd       }
397                            1035,
398                            4538,  {                 }
399     {  2295  }               3,  {  Merge           }
400                            2300,  {                 }
401     {  2297  }              14,  {  oEmitLine       }
402     {  2298  }               2,  {  Repeat          }
403                            2279,  {                 }
404     {  2300  }               3,  {  Merge           }
405                            2305,  {                 }
406     {  2302  }               1,  {  TABLE BEGINS    }
407                              14,  {  tCodeStart     }
408                            2279,
409     {  2305  }               4,  {  Input           }
410                              65,  {  tEnd           }
411                            1041,
412                            3554,  {                 }
413     {  2309  }               6,  {  Emit            }
414                              42,  {  aEndCode       }
415     {  2311  }               5,  {  InputChoice     }
416                            2322,  {  TABLE          }
417                            1042,
418                            3554,  {                 }
419     {  2315  }               6,  {  Emit            }
420                              46,  {  aEndIdent      }
421     {  2317  }               6,  {  Emit            }
422                               0,  {  aIdent         }
423     {  2319  }               9,  {  oEmitIdent      }
424     {  2320  }               3,  {  Merge           }
425                            2325,  {                 }
426     {  2322  }               1,  {  TABLE BEGINS    }
427                               0,  {  tIdent         }
428                            2315,
429     {  2325  }               3,  {  Merge           }
430                            2346,  {                 }
431     {  2327  }               6,  {  Emit            }
432                              66,  {  aForward       }
433     {  2329  }               3,  {  Merge           }
434                            2346,  {                 }
435     {  2331  }               6,  {  Emit            }
436                              62,  {  aExternal      }
437     {  2333  }               3,  {  Merge           }
438                            2346,  {                 }
439     {  2335  }               4,  {  TABLE BEGINS    }
440                              47,  {  tBegin         }
441                            2245,
442                              54,  {  tCode          }
443                            2272,
444                              71,  {  tForward       }
```

```
445                          2327,
446                            68,  {                    tExternal       }
447                          2331,
448        {   2344   }        2,  {                    Repeat          }
449                          2245,  {                                    }
450        {   2346   }        1,  {                    Return          }
451        {   2347   }        5,  { Statements         InputChoice     }
452                          2398,  {                    TABLE           }
453                          1058,
454                          4570,  {                                    }
455        {   2351   }        0,  {                    Call            }
456                          2433,  {                    AssignmentOrCal }
457                          4570,  {                                    }
458        {   2354   }        3,  {                    Merge           }
459                          2425,  {                                    }
460        {   2356   }        0,  {                    Call            }
461                          2451,  {                    ExitStatement   }
462                          4570,  {                                    }
463        {   2359   }        3,  {                    Merge           }
464                          2425,  {                                    }
465        {   2361   }        0,  {                    Call            }
466                          2468,  {                    ReturnStatement }
467                          4570,  {                                    }
468        {   2364   }        3,  {                    Merge           }
469                          2425,  {                                    }
470        {   2366   }        0,  {                    Call            }
471                          2503,  {                    AssertStatement }
472                          4570,  {                                    }
473        {   2369   }        3,  {                    Merge           }
474                          2425,  {                                    }
475        {   2371   }        0,  {                    Call            }
476                          2522,  {                    BeginStatement  }
477                          4570,  {                                    }
478        {   2374   }        3,  {                    Merge           }
479                          2425,  {                                    }
480        {   2376   }        0,  {                    Call            }
481                          2573,  {                    IfStatement     }
482                          4570,  {                                    }
483        {   2379   }        3,  {                    Merge           }
484                          2425,  {                                    }
485        {   2381   }        0,  {                    Call            }
486                          2651,  {                    LoopStatement   }
487                          4570,  {                                    }
488        {   2384   }        3,  {                    Merge           }
489                          2425,  {                                    }
490        {   2386   }        0,  {                    Call            }
491                          2671,  {                    ForLoopStatemen }
492                          4570,  {                                    }
493        {   2389   }        3,  {                    Merge           }
494                          2425,  {                                    }
495        {   2391   }        0,  {                    Call            }
496                          2723,  {                    CaseStatement   }
```

```
497                          4570,  {                                 }
498     {  2394  }              3,  {  Merge                          }
499                          2425,  {                                 }
500     {  2396  }              3,  {  Merge                          }
501                          2425,  {                                 }
502     {  2398  }             10,  {  TABLE BEGINS                   }
503                             0,  {  tIdent                         }
504                          2351,
505                            66,  {  tExit                          }
506                          2356,
507                            97,  {  tReturn                        }
508                          2361,
509                            45,  {  tAssert                        }
510                          2366,
511                            47,  {  tBegin                         }
512                          2371,
513                            74,  {  tIf                            }
514                          2376,
515                            81,  {  tLoop                          }
516                          2381,
517                            70,  {  tFor                           }
518                          2386,
519                            51,  {  tCase                          }
520                          2391,
521                            35,  {  tSemicolon                     }
522                          2396,
523     {  2419  }              4,  {  Input                          }
524                            35,  {  tSemicolon                     }
525                          1079,
526                          3562,  {                                 }
527     {  2423  }              3,  {  Merge                          }
528                          2432,  {                                 }
529     {  2425  }              4,  {  Input                          }
530                            35,  {  tSemicolon                     }
531                          1081,
532                          3570,  {                                 }
533     {  2429  }             14,  {  oEmitLine                      }
534     {  2430  }              2,  {  Repeat                         }
535                          2347,  {                                 }
536     {  2432  }              1,  {  Return                         }
537     {  2433  }              0,  {  AssignmentOrCal  Call          }
538                          3407,  {  Variable                       }
539                          3882,  {                                 }
540     {  2436  }              5,  {  InputChoice                    }
541                          2447,  {  TABLE                          }
542                          1089,
543                          3882,  {                                 }
544     {  2440  }              6,  {  Emit                           }
545                            19,  {  aAssign                        }
546     {  2442  }              0,  {  Call                           }
547                          3000,  {  Expression                     }
548                          3570,  {                                 }
```

```
549     {  2445  }     3,  {                    Merge          }
550                 2450,  {                                   }
551     {  2447  }     1,  {                    TABLE BEGINS   }
552                   33,  {                    tAssign        }
553                 2440,
554     {  2450  }     1,  {                    Return         }
555     {  2451  }     6,  { ExitStatement      Emit           }
556                   60,  {                    aExit          }
557     {  2453  }     5,  {                    InputChoice    }
558                 2464,  {                    TABLE          }
559                 1100,
560                 4634,  {                                   }
561     {  2457  }     6,  {                    Emit           }
562                  124,  {                    aWhen          }
563     {  2459  }     0,  {                    Call           }
564                 3000,  {                    Expression     }
565                 3570,  {                                   }
566     {  2462  }     3,  {                    Merge          }
567                 2467,  {                                   }
568     {  2464  }     1,  {                    TABLE BEGINS   }
569                  106,  {                    tWhen          }
570                 2457,
571     {  2467  }     1,  {                    Return         }
572     {  2468  }     6,  { ReturnStatement    Emit           }
573                  111,  {                    aReturn        }
574     {  2470  }     5,  {                    InputChoice    }
575                 2485,  {                    TABLE          }
576                 1111,
577                 4642,  {                                   }
578     {  2474  }     6,  {                    Emit           }
579                  112,  {                    aReturnValue   }
580     {  2476  }     0,  {                    Call           }
581                 3000,  {                    Expression     }
582                 4642,  {                                   }
583     {  2479  }     4,  {                    Input          }
584                   30,  {                    tRightParen    }
585                 1113,
586                 4634,  {                                   }
587     {  2483  }     3,  {                    Merge          }
588                 2488,  {                                   }
589     {  2485  }     1,  {                    TABLE BEGINS   }
590                   29,  {                    tLeftParen     }
591                 2474,
592     {  2488  }     5,  {                    InputChoice    }
593                 2499,  {                    TABLE          }
594                 1116,
595                 4634,  {                                   }
596     {  2492  }     6,  {                    Emit           }
597                  124,  {                    aWhen          }
598     {  2494  }     0,  {                    Call           }
599                 3000,  {                    Expression     }
600                 3570,  {                                   }
```

```
601     {  2497  }     3,  {                    Merge          }
602                 2502,  {                                   }
603     {  2499  }     1,  {                    TABLE BEGINS   }
604                  106,  {                    tWhen          }
605                 2492,
606     {  2502  }     1,  {                    Return         }
607     {  2503  }     5,  { AssertStatement    InputChoice    }
608                 2518,  {                    TABLE          }
609                 1126,
610                 4650,  {                                   }
611     {  2507  }     6,  {                    Emit           }
612                   18,  {                    aAssert        }
613     {  2509  }     0,  {                    Call           }
614                 3000,  {                    Expression     }
615                 3978,  {                                   }
616     {  2512  }     4,  {                    Input          }
617                   30,  {                    tRightParen    }
618                 1128,
619                 3570,  {                                   }
620     {  2516  }     3,  {                    Merge          }
621                 2521,  {                                   }
622     {  2518  }     1,  {                    TABLE BEGINS   }
623                   29,  {                    tLeftParen     }
624                 2507,
625     {  2521  }     1,  {                    Return         }
626     {  2522  }     6,  { BeginStatement     Emit           }
627                   21,  {                    aBegin         }
628     {  2524  }     0,  {                    Call           }
629                 2534,  {                    BlockBody      }
630                 4138,  {                                   }
631     {  2527  }     4,  {                    Input          }
632                   65,  {                    tEnd           }
633                 1138,
634                 3570,  {                                   }
635     {  2531  }     6,  {                    Emit           }
636                   39,  {                    aEndBegin      }
637     {  2533  }     1,  {                    Return         }
638     {  2534  }    14,  { BlockBody          oEmitLine      }
639     {  2535  }     5,  {                    InputChoice    }
640                 2561,  {                    TABLE          }
641                 1145,
642                 4658,  {                                   }
643     {  2539  }     4,  {                    Input          }
644                   53,  {                    tChecked       }
645                 1147,
646                 4666,  {                                   }
647     {  2543  }     6,  {                    Emit           }
648                   94,  {                    aNotChecked    }
649     {  2545  }     4,  {                    Input          }
650                   35,  {                    tSemicolon     }
651                 1147,
652                 4666,  {                                   }
```

```
653     {  2549  }     14,  {                    oEmitLine      }
654     {  2550  }      3,  {                    Merge          }
655                  2566,  {                                   }
656     {  2552  }      6,  {                    Emit           }
657                    27,  {                    aChecked       }
658     {  2554  }      4,  {                    Input          }
659                    35,  {                    tSemicolon     }
660                  1150,
661                  4666,  {                                   }
662     {  2558  }     14,  {                    oEmitLine      }
663     {  2559  }      3,  {                    Merge          }
664                  2566,  {                                   }
665     {  2561  }      2,  {                    TABLE BEGINS   }
666                    85,  {                    tNot           }
667                  2539,
668                    53,  {                    tChecked       }
669                  2552,
670     {  2566  }      0,  {                    Call           }
671                   488,  {                    Declarations   }
672                  4562,  {                                   }
673     {  2569  }      0,  {                    Call           }
674                  2347,  {                    Statements     }
675                  3570,  {                                   }
676     {  2572  }      1,  {                    Return         }
677     {  2573  }      6,  { IfStatement        Emit           }
678                    70,  {                    aIf            }
679     {  2575  }      0,  {                    Call           }
680                  3000,  {                    Expression     }
681                  4674,  {                                   }
682     {  2578  }      4,  {                    Input          }
683                   100,  {                    tThen          }
684                  1161,
685                  4682,  {                                   }
686     {  2582  }      4,  {                    Input          }
687                    18,  {                    tNewLine       }
688                  1161,
689                  4690,  {                                   }
690     {  2586  }      0,  {                    Call           }
691                  2534,  {                    BlockBody      }
692                  4698,  {                                   }
693     {  2589  }      0,  {                    Call           }
694                  2603,  {                    ElseClause     }
695                  4714,  {                                   }
696     {  2592  }      4,  {                    Input          }
697                    65,  {                    tEnd           }
698                  1164,
699                  4722,  {                                   }
700     {  2596  }      4,  {                    Input          }
701                    74,  {                    tIf            }
702                  1164,
703                  3570,  {                                   }
704     {  2600  }      6,  {                    Emit           }
```

```
705                          47,  {                        aEndIf          }
706     {  2602  }            1,  {                        Return          }
707     {  2603  }            5,  { ElseClause             InputChoice     }
708                        2645,  {                        TABLE           }
709                        1170,
710                        4730,  {                                        }
711     {  2607  }            6,  {                        Emit            }
712                          38,  {                        aElse           }
713     {  2609  }           14,  {                        oEmitLine       }
714     {  2610  }            6,  {                        Emit            }
715                          70,  {                        aIf             }
716     {  2612  }            0,  {                        Call            }
717                        3000,  {                        Expression      }
718                        4738,  {                                        }
719     {  2615  }            4,  {                        Input           }
720                         100,  {                        tThen           }
721                        1173,
722                        4746,  {                                        }
723     {  2619  }            4,  {                        Input           }
724                          18,  {                        tNewLine        }
725                        1173,
726                        4754,  {                                        }
727     {  2623  }            0,  {                        Call            }
728                        2534,  {                        BlockBody       }
729                        4706,  {                                        }
730     {  2626  }            0,  {                        Call            }
731                        2603,  {                        ElseClause      }
732                        3570,  {                                        }
733     {  2629  }            6,  {                        Emit            }
734                          47,  {                        aEndIf          }
735     {  2631  }           14,  {                        oEmitLine       }
736     {  2632  }            3,  {                        Merge           }
737                        2650,  {                                        }
738     {  2634  }            6,  {                        Emit            }
739                          38,  {                        aElse           }
740     {  2636  }            4,  {                        Input           }
741                          18,  {                        tNewLine        }
742                        1178,
743                        4530,  {                                        }
744     {  2640  }            0,  {                        Call            }
745                        2534,  {                        BlockBody       }
746                        3570,  {                                        }
747     {  2643  }            3,  {                        Merge           }
748                        2650,  {                                        }
749     {  2645  }            2,  {                        TABLE BEGINS    }
750                          64,  {                        tElseif         }
751                        2607,
752                          63,  {                        tElse           }
753                        2634,
754     {  2650  }            1,  {                        Return          }
755     {  2651  }            6,  { LoopStatement          Emit            }
756                          85,  {                        aLoop           }
```

```
757     {  2653  }      4,  {                       Input          }
758                    18,  {                       tNewLine       }
759                  1187,
760                  4762,  {                                      }
761     {  2657  }      0,  {                       Call           }
762                  2534,  {                       BlockBody      }
763                  4770,  {                                      }
764     {  2660  }      4,  {                       Input          }
765                    65,  {                       tEnd           }
766                  1189,
767                  4778,  {                                      }
768     {  2664  }      4,  {                       Input          }
769                    81,  {                       tLoop          }
770                  1189,
771                  3570,  {                                      }
772     {  2668  }      6,  {                       Emit           }
773                    51,  {                       aEndLoop       }
774     {  2670  }      1,  {                       Return         }
775     {  2671  }      6,  { ForLoopStatemen       Emit           }
776                    65,  {                       aFor           }
777     {  2673  }      4,  {                       Input          }
778                     0,  {                       tIdent         }
779                  1195,
780                  4786,  {                                      }
781     {  2677  }      6,  {                       Emit           }
782                     0,  {                       aIdent         }
783     {  2679  }      9,  {                       oEmitIdent     }
784     {  2680  }      5,  {                       InputChoice    }
785                  2688,  {                       TABLE          }
786                  1196,
787                  4786,  {                                      }
788     {  2684  }      6,  {                       Emit           }
789                    35,  {                       aDecreasing    }
790     {  2686  }      3,  {                       Merge          }
791                  2691,  {                                      }
792     {  2688  }      1,  {                       TABLE BEGINS   }
793                    59,  {                       tDecreasing    }
794                  2684,
795     {  2691  }      4,  {                       Input          }
796                    76,  {                       tIn            }
797                  1201,
798                  4794,  {                                      }
799     {  2695  }      6,  {                       Emit           }
800                    73,  {                       aIn            }
801     {  2697  }      0,  {                       Call           }
802                  3000,  {                       Expression     }
803                  4802,  {                                      }
804     {  2700  }      5,  {                       InputChoice    }
805                  2711,  {                       TABLE          }
806                  1202,
807                  4802,  {                                      }
808     {  2704  }      6,  {                       Emit           }
```

```
809                          117,  {                    aTo          }
810     {  2706  }             0,  {                    Call         }
811                           3224,  {                    Sum          }
812                           4810,  {                                 }
813     {  2709  }             3,  {                    Merge        }
814                           2714,  {                                 }
815     {  2711  }             1,  {                    TABLE BEGINS }
816                            32,  {                    tDoublePeriod }
817                           2704,
818     {  2714  }            14,  {                    oEmitLine    }
819     {  2715  }             4,  {                    Input        }
820                            81,  {                    tLoop        }
821                           1208,
822                           4618,  {                                 }
823     {  2719  }             0,  {                    Call         }
824                           2651,  {                    LoopStatement }
825                           3570,  {                                 }
826     {  2722  }             1,  {                    Return       }
827     {  2723  }             6,  { CaseStatement      Emit         }
828                            25,  {                    aCase        }
829     {  2725  }             5,  {                    InputChoice  }
830                           2862,  {                    TABLE        }
831                           1215,
832                           4818,  {                                 }
833     {  2729  }             6,  {                    Emit         }
834                           122,  {                    aVar         }
835     {  2731  }             4,  {                    Input        }
836                             0,  {                    tIdent       }
837                           1217,
838                           4826,  {                                 }
839     {  2735  }             6,  {                    Emit         }
840                             0,  {                    aIdent       }
841     {  2737  }             9,  {                    oEmitIdent   }
842     {  2738  }             4,  {                    Input        }
843                            50,  {                    tBound       }
844                           1217,
845                           4834,  {                                 }
846     {  2742  }             4,  {                    Input        }
847                           102,  {                    tTo          }
848                           1217,
849                           4842,  {                                 }
850     {  2746  }             6,  {                    Emit         }
851                            24,  {                    aBound       }
852     {  2748  }             4,  {                    Input        }
853                             0,  {                    tIdent       }
854                           1218,
855                           4842,  {                                 }
856     {  2752  }             0,  {                    Call         }
857                           3407,  {                    Variable     }
858                           4850,  {                                 }
859     {  2755  }             3,  {                    Merge        }
860                           2874,  {                                 }
```

```
861     {  2757  }      6,  {                   Emit          }
862                   109,  {                   aReadonly     }
863     {  2759  }      4,  {                   Input         }
864                     0,  {                   tIdent        }
865                  1220,
866                  4826,  {                                 }
867     {  2763  }      6,  {                   Emit          }
868                     0,  {                   aIdent        }
869     {  2765  }      9,  {                   oEmitIdent    }
870     {  2766  }      4,  {                   Input         }
871                    50,  {                   tBound        }
872                  1220,
873                  4834,  {                                 }
874     {  2770  }      4,  {                   Input         }
875                   102,  {                   tTo           }
876                  1220,
877                  4842,  {                                 }
878     {  2774  }      6,  {                   Emit          }
879                    24,  {                   aBound        }
880     {  2776  }      4,  {                   Input         }
881                     0,  {                   tIdent        }
882                  1221,
883                  4842,  {                                 }
884     {  2780  }      0,  {                   Call          }
885                  3407,  {                   Variable      }
886                  4850,  {                                 }
887     {  2783  }      3,  {                   Merge         }
888                  2874,  {                                 }
889     {  2785  }      6,  {                   Emit          }
890                    30,  {                   aConst        }
891     {  2787  }      4,  {                   Input         }
892                     0,  {                   tIdent        }
893                  1223,
894                  4858,  {                                 }
895     {  2791  }      6,  {                   Emit          }
896                     0,  {                   aIdent        }
897     {  2793  }      9,  {                   oEmitIdent    }
898     {  2794  }      4,  {                   Input         }
899                    33,  {                   tAssign       }
900                  1223,
901                  4842,  {                                 }
902     {  2798  }      6,  {                   Emit          }
903                    19,  {                   aAssign       }
904     {  2800  }      4,  {                   Input         }
905                     0,  {                   tIdent        }
906                  1223,
907                  4842,  {                                 }
908     {  2804  }      0,  {                   Call          }
909                  3407,  {                   Variable      }
910                  4850,  {                                 }
911     {  2807  }      3,  {                   Merge         }
912                  2874,  {                                 }
```

```
913     {  2809  }      5,  {   InputChoice     }
914                    2849,  {   TABLE          }
915                    1225,
916                    4866,  {                  }
917     {  2813  }      6,  {   Emit             }
918                      30,  {   aConst          }
919     {  2815  }      6,  {   Emit             }
920                       0,  {   aIdent          }
921     {  2817  }      9,  {   oEmitIdent       }
922     {  2818  }      6,  {   Emit             }
923                      19,  {   aAssign         }
924     {  2820  }      4,  {   Input            }
925                       0,  {   tIdent          }
926                    1227,
927                    4842,  {                  }
928     {  2824  }      0,  {   Call             }
929                    3407,  {   Variable       }
930                    4850,  {                  }
931     {  2827  }      3,  {   Merge            }
932                    2860,  {                  }
933     {  2829  }      4,  {   Input            }
934                     102,  {   tTo            }
935                    1229,
936                    4842,  {                  }
937     {  2833  }      6,  {   Emit             }
938                     109,  {   aReadonly      }
939     {  2835  }      6,  {   Emit             }
940                       0,  {   aIdent          }
941     {  2837  }      9,  {   oEmitIdent       }
942     {  2838  }      6,  {   Emit             }
943                      24,  {   aBound          }
944     {  2840  }      4,  {   Input            }
945                       0,  {   tIdent          }
946                    1230,
947                    4842,  {                  }
948     {  2844  }      0,  {   Call             }
949                    3407,  {   Variable       }
950                    4850,  {                  }
951     {  2847  }      3,  {   Merge            }
952                    2860,  {                  }
953     {  2849  }      2,  {   TABLE BEGINS     }
954                      33,  {   tAssign         }
955                    2813,
956                      50,  {   tBound          }
957                    2829,
958     {  2854  }      0,  {   Call             }
959                    3407,  {   Variable       }
960                    4850,  {                  }
961     {  2857  }      0,  {   Call             }
962                    3237,  {   PartialSum     }
963                    4850,  {                  }
964     {  2860  }      3,  {   Merge            }
```

```
965                          2874,  {                                 }
966     {  2862  }             4,   {  TABLE BEGINS                    }
967                          105,   {  tVar                            }
968                         2729,
969                           95,   {  tReadonly                       }
970                         2757,
971                           56,   {  tConst                          }
972                         2785,
973                            0,   {  tIdent                          }
974                         2809,
975     {  2871  }             0,   {  Call                             }
976                         3224,   {  Sum                              }
977                         4850,   {                                   }
978     {  2874  }             4,   {  Input                            }
979                           86,   {  tOf                              }
980                         1237,
981                         4874,   {                                   }
982     {  2878  }             4,   {  Input                            }
983                           18,   {  tNewLine                         }
984                         1237,
985                         4882,   {                                   }
986     {  2882  }             0,   {  Call                             }
987                         2888,   {  CaseBody                         }
988                         3570,   {                                   }
989     {  2885  }             6,   {  Emit                             }
990                           41,   {  aEndCase                         }
991     {  2887  }             1,   {  Return                           }
992     {  2888  }            14,   {  CaseBody     oEmitLine          }
993     {  2889  }             5,   {  InputChoice                      }
994                         2926,   {  TABLE                            }
995                         1245,
996                         4890,   {                                   }
997     {  2893  }             4,   {  Input                            }
998                           51,   {  tCase                            }
999                         1247,
000                         3570,   {                                   }
001     {  2897  }             3,   {  Merge                            }
002                         2981,   {                                   }
003     {  2899  }             3,   {  Merge                            }
004                         2979,   {                                   }
005     {  2901  }             4,   {  Input                            }
006                           38,   {  tLabel                           }
007                         1249,
008                         4898,   {                                   }
009     {  2905  }             4,   {  Input                            }
010                           18,   {  tNewLine                         }
011                         1249,
012                         4906,   {                                   }
013     {  2909  }             6,   {  Emit                             }
014                           98,   {  aOtherwise                       }
015     {  2911  }             0,   {  Call                             }
016                         2534,   {  BlockBody                        }
```

```
017                          4914,  {                          }
018     {   2914   }           4,  {   Input                    }
019                            65,  {   tEnd                    }
020                          1251,
021                          4922,  {                          }
022     {   2918   }           4,  {   Input                    }
023                            51,  {   tCase                   }
024                          1251,
025                          3570,  {                          }
026     {   2922   }           3,  {   Merge                    }
027                          2981,  {                          }
028     {   2924   }           3,  {   Merge                    }
029                          2979,  {                          }
030     {   2926   }           2,  {   TABLE BEGINS             }
031                            65,  {   tEnd                    }
032                          2893,
033                            88,  {   tOtherwise              }
034                          2901,
035     {   2931   }           6,  {   Emit                     }
036                            82,  {   aLabels                 }
037     {   2933   }           0,  {   Call                     }
038                          2982,  {   CaseLabel               }
039                          4890,  {                          }
040     {   2936   }           5,  {   InputChoice              }
041                          2945,  {   TABLE                  }
042                          1254,
043                          4890,  {                          }
044     {   2940   }           0,  {   Call                     }
045                          2982,  {   CaseLabel               }
046                          4890,  {                          }
047     {   2943   }           3,  {   Merge                    }
048                          2950,  {                          }
049     {   2945   }           1,  {   TABLE BEGINS             }
050                            34,  {   tComma                  }
051                          2940,
052     {   2948   }           3,  {   Merge                    }
053                          2952,  {                          }
054     {   2950   }           2,  {   Repeat                   }
055                          2936,  {                          }
056     {   2952   }           6,  {   Emit                     }
057                            50,  {   aEndLabels              }
058     {   2954   }           4,  {   Input                    }
059                            38,  {   tLabel                  }
060                          1260,
061                          4890,  {                          }
062     {   2958   }           4,  {   Input                    }
063                            18,  {   tNewLine                }
064                          1260,
065                          4890,  {                          }
066     {   2962   }           0,  {   Call                     }
067                          2534,  {   BlockBody               }
068                          4890,  {                          }
```

```
069         {   2965   }        4,  {                       Input          }
070                              65,  {                       tEnd          }
071                            1262,
072                            4890,  {                                     }
073         {   2969   }        6,  {                       Emit           }
074                              49,  {                       aEndLabel      }
075         {   2971   }         0,  {                       Call           }
076                            2982,  {                       CaseLabel      }
077                            4890,  {                                     }
078         {   2974   }        4,  {                       Input          }
079                              35,  {                       tSemicolon     }
080                            1262,
081                            4890,  {                                     }
082         {   2978   }        14,  {                       oEmitLine      }
083         {   2979   }         2,  {                       Repeat         }
084                            2889,  {                                     }
085         {   2981   }         1,  {                       Return         }
086         {   2982   }         0,  { CaseLabel             Call           }
087                            3000,  {                       Expression     }
088                            4098,  {                                     }
089         {   2985   }        5,  {                       InputChoice    }
090                            2996,  {                       TABLE          }
091                            1271,
092                            4098,  {                                     }
093         {   2989   }        6,  {                       Emit           }
094                             117,  {                       aTo            }
095         {   2991   }         0,  {                       Call           }
096                            3224,  {                       Sum            }
097                            3570,  {                                     }
098         {   2994   }         3,  {                       Merge          }
099                            2999,  {                                     }
100         {   2996   }         1,  {                       TABLE BEGINS   }
101                              32,  {                       tDoublePeriod  }
102                            2989,
103         {   2999   }         1,  {                       Return         }
104         {   3000   }         0,  { Expression            Call           }
105                            3006,  {                       SubExpression  }
106                            3570,  {                                     }
107         {   3003   }         6,  {                       Emit           }
108                              45,  {                       aEndExpression }
109         {   3005   }         1,  {                       Return         }
110         {   3006   }         0,  { SubExpression         Call           }
111                            3026,  {                       Disjunction    }
112                            3682,  {                                     }
113         {   3009   }        5,  {                       InputChoice    }
114                            3022,  {                       TABLE          }
115                            1288,
116                            3682,  {                                     }
117         {   3013   }         6,  {                       Emit           }
118                              76,  {                       aInfixImply    }
119         {   3015   }         0,  {                       Call           }
120                            3026,  {                       Disjunction    }
```

```
121                          3570,  {                                }
122     {  3018  }             6,   {  Emit                           }
123                           71,   {  aImply                         }
124     {  3020  }             3,   {  Merge                          }
125                          3025,  {                                }
126     {  3022  }             1,   {  TABLE BEGINS                   }
127                           28,   {  tImplies                       }
128                          3013,
129     {  3025  }             1,   {  Return                         }
130     {  3026  }             0,   {  Disjunction     Call           }
131                          3050,  {  Conjunction                   }
132                          4930,  {                                }
133     {  3029  }             5,   {  InputChoice                    }
134                          3042,  {  TABLE                         }
135                          1299,
136                          4930,  {                                }
137     {  3033  }             6,   {  Emit                           }
138                           77,   {  aInfixOr                       }
139     {  3035  }             0,   {  Call                           }
140                          3050,  {  Conjunction                   }
141                          4930,  {                                }
142     {  3038  }             6,   {  Emit                           }
143                           97,   {  aOr                            }
144     {  3040  }             3,   {  Merge                          }
145                          3047,  {                                }
146     {  3042  }             1,   {  TABLE BEGINS                   }
147                           87,   {  tOr                            }
148                          3033,
149     {  3045  }             3,   {  Merge                          }
150                          3049,  {                                }
151     {  3047  }             2,   {  Repeat                         }
152                          3029,  {                                }
153     {  3049  }             1,   {  Return                         }
154     {  3050  }             0,   {  Conjunction     Call           }
155                          3074,  {  Negation                      }
156                          4938,  {                                }
157     {  3053  }             5,   {  InputChoice                    }
158                          3066,  {  TABLE                         }
159                          1311,
160                          4938,  {                                }
161     {  3057  }             6,   {  Emit                           }
162                           74,   {  aInfixAnd                      }
163     {  3059  }             0,   {  Call                           }
164                          3074,  {  Negation                      }
165                          4938,  {                                }
166     {  3062  }             6,   {  Emit                           }
167                           15,   {  aAnd                           }
168     {  3064  }             3,   {  Merge                          }
169                          3071,  {                                }
170     {  3066  }             1,   {  TABLE BEGINS                   }
171                           42,   {  tAnd                           }
172                          3057,
```

```
173        {   3069   }        3,  {                        Merge          }
174                          3073,  {                                       }
175        {   3071   }        2,  {                        Repeat         }
176                          3053,  {                                       }
177        {   3073   }        1,  {                        Return         }
178        {   3074   }        5,  { Negation               InputChoice    }
179                          3085,  {                        TABLE          }
180                          1322,
181                          4946,  {                                       }
182        {   3078   }        0,  {                        Call           }
183                          3092,  {                        Relation       }
184                          3570,  {                                       }
185        {   3081   }        6,  {                        Emit           }
186                            93,  {                        aNot           }
187        {   3083   }        3,  {                        Merge          }
188                          3091,  {                                       }
189        {   3085   }        1,  {                        TABLE BEGINS   }
190                            85,  {                        tNot           }
191                          3078,
192        {   3088   }        0,  {                        Call           }
193                          3092,  {                        Relation       }
194                          3570,  {                                       }
195        {   3091   }        1,  {                        Return         }
196        {   3092   }        0,  { Relation               Call           }
197                          3230,  {                        SubSum         }
198                          4954,  {                                       }
199        {   3095   }        5,  {                        InputChoice    }
200                          3208,  {                        TABLE          }
201                          1334,
202                          4954,  {                                       }
203        {   3099   }        6,  {                        Emit           }
204                            75,  {                        aInfixCompare  }
205        {   3101   }        0,  {                        Call           }
206                          3230,  {                        SubSum         }
207                          3570,  {                                       }
208        {   3104   }        6,  {                        Emit           }
209                            59,  {                        aEqual         }
210        {   3106   }        3,  {                        Merge          }
211                          3223,  {                                       }
212        {   3108   }        6,  {                        Emit           }
213                            75,  {                        aInfixCompare  }
214        {   3110   }        0,  {                        Call           }
215                          3230,  {                        SubSum         }
216                          3570,  {                                       }
217        {   3113   }        6,  {                        Emit           }
218                            83,  {                        aLess          }
219        {   3115   }        3,  {                        Merge          }
220                          3223,  {                                       }
221        {   3117   }        6,  {                        Emit           }
222                            75,  {                        aInfixCompare  }
223        {   3119   }        0,  {                        Call           }
224                          3230,  {                        SubSum         }
```

```
225                          3570,  {                                }
226     {   3122   }            6,  {  Emit                          }
227                             84,  {  aLessEqual                    }
228     {   3124   }            3,  {  Merge                         }
229                          3223,  {                                }
230     {   3126   }            6,  {  Emit                          }
231                             75,  {  aInfixCompare                 }
232     {   3128   }            0,  {  Call                          }
233                          3230,  {  SubSum                        }
234                          3570,  {                                }
235     {   3131   }            6,  {  Emit                          }
236                             68,  {  aGreater                      }
237     {   3133   }            3,  {  Merge                         }
238                          3223,  {                                }
239     {   3135   }            6,  {  Emit                          }
240                             75,  {  aInfixCompare                 }
241     {   3137   }            0,  {  Call                          }
242                          3230,  {  SubSum                        }
243                          3570,  {                                }
244     {   3140   }            6,  {  Emit                          }
245                             69,  {  aGreaterEqual                 }
246     {   3142   }            3,  {  Merge                         }
247                          3223,  {                                }
248     {   3144   }            5,  {  InputChoice                   }
249                          3178,  {  TABLE                         }
250                          1346,
251                          4962,  {                                }
252     {   3148   }            6,  {  Emit                          }
253                             75,  {  aInfixCompare                 }
254     {   3150   }            0,  {  Call                          }
255                          3230,  {  SubSum                        }
256                          3570,  {                                }
257     {   3153   }            6,  {  Emit                          }
258                             95,  {  aNotEqual                     }
259     {   3155   }            3,  {  Merge                         }
260                          3185,  {                                }
261     {   3157   }            6,  {  Emit                          }
262                             96,  {  aNotIn                        }
263     {   3159   }            0,  {  Call                          }
264                          3224,  {  Sum                           }
265                          4098,  {                                }
266     {   3162   }            5,  {  InputChoice                   }
267                          3173,  {  TABLE                         }
268                          1351,
269                          4098,  {                                }
270     {   3166   }            6,  {  Emit                          }
271                            117,  {  aTo                          }
272     {   3168   }            0,  {  Call                          }
273                          3224,  {  Sum                           }
274                          3570,  {                                }
275     {   3171   }            3,  {  Merge                         }
276                          3176,  {                                }
```

```
277     {  3173  }     1,  {                    TABLE BEGINS    }
278                   32,  {                    tDoublePeriod   }
279                 3166,
280     {  3176  }     3,  {                    Merge           }
281                 3185,  {                                    }
282     {  3178  }     2,  {                    TABLE BEGINS    }
283                   23,  {                    tEqual          }
284                 3148,
285                   76,  {                    tIn             }
286                 3157,
287     {  3183  }     2,  {                    Repeat          }
288                 3148,  {                                    }
289     {  3185  }     3,  {                    Merge           }
290                 3223,  {                                    }
291     {  3187  }     6,  {                    Emit            }
292                   73,  {                    aIn             }
293     {  3189  }     0,  {                    Call            }
294                 3224,  {                    Sum             }
295                 4098,  {                                    }
296     {  3192  }     5,  {                    InputChoice     }
297                 3203,  {                    TABLE           }
298                 1359,
299                 4098,  {                                    }
300     {  3196  }     6,  {                    Emit            }
301                  117,  {                    aTo             }
302     {  3198  }     0,  {                    Call            }
303                 3224,  {                    Sum             }
304                 3570,  {                                    }
305     {  3201  }     3,  {                    Merge           }
306                 3206,  {                                    }
307     {  3203  }     1,  {                    TABLE BEGINS    }
308                   32,  {                    tDoublePeriod   }
309                 3196,
310     {  3206  }     3,  {                    Merge           }
311                 3223,  {                                    }
312     {  3208  }     7,  {                    TABLE BEGINS    }
313                   23,  {                    tEqual          }
314                 3099,
315                   24,  {                    tLessThan       }
316                 3108,
317                   26,  {                    tLessEqual      }
318                 3117,
319                   25,  {                    tGreaterThan    }
320                 3126,
321                   27,  {                    tGreaterEqual   }
322                 3135,
323                   85,  {                    tNot            }
324                 3144,
325                   76,  {                    tIn             }
326                 3187,
327     {  3223  }     1,  {                    Return          }
328     {  3224  }     0,  { Sum                Call            }
```

```
329                          3230,  {                    SubSum          }
330                          3570,  {                                    }
331        {  3227  }           6,  {                    Emit            }
332                             45,  {                    aEndExpression  }
333        {  3229  }           1,  {                    Return          }
334        {  3230  }           0,  { SubSum              Call            }
335                          3327,  {                    Factor          }
336                          4258,  {                                    }
337        {  3233  }           0,  {                    Call            }
338                          3243,  {                    PartialSubSum   }
339                          3570,  {                                    }
340        {  3236  }           1,  {                    Return          }
341        {  3237  }           0,  { PartialSum          Call            }
342                          3243,  {                    PartialSubSum   }
343                          3570,  {                                    }
344        {  3240  }           6,  {                    Emit            }
345                             45,  {                    aEndExpression  }
346        {  3242  }           1,  {                    Return          }
347        {  3243  }           0,  { PartialSubSum       Call            }
348                          3290,  {                    PartialTerm     }
349                          4274,  {                                    }
350        {  3246  }           5,  {                    InputChoice     }
351                          3271,  {                    TABLE           }
352                          1390,
353                          4274,  {                                    }
354        {  3250  }           0,  {                    Call            }
355                          3283,  {                    Term            }
356                          4274,  {                                    }
357        {  3253  }           6,  {                    Emit            }
358                             12,  {                    aAdd            }
359        {  3255  }           3,  {                    Merge           }
360                          3280,  {                                    }
361        {  3257  }           0,  {                    Call            }
362                          3283,  {                    Term            }
363                          4274,  {                                    }
364        {  3260  }           6,  {                    Emit            }
365                            116,  {                    aSubtract       }
366        {  3262  }           3,  {                    Merge           }
367                          3280,  {                                    }
368        {  3264  }           0,  {                    Call            }
369                          3283,  {                    Term            }
370                          4274,  {                                    }
371        {  3267  }           6,  {                    Emit            }
372                            126,  {                    aXor            }
373        {  3269  }           3,  {                    Merge           }
374                          3280,  {                                    }
375        {  3271  }           3,  {                    TABLE BEGINS    }
376                             20,  {                    tPlus           }
377                          3250,
378                             21,  {                    tMinus          }
379                          3257,
380                            108,  {                    tXor            }
```

```
381                          3264,
382     {  3278  }              3,  {                    Merge       }
383                          3282,  {                                 }
384     {  3280  }              2,  {                    Repeat      }
385                          3246,  {                                 }
386     {  3282  }              1,  {                    Return      }
387     {  3283  }              0,  { Term               Call        }
388                          3327,  {                    Factor      }
389                          4970,  {                                 }
390     {  3286  }              0,  {                    Call        }
391                          3290,  {                    PartialTerm }
392                          3570,  {                                 }
393     {  3289  }              1,  {                    Return      }
394     {  3290  }              5,  { PartialTerm        InputChoice }
395                          3315,  {                    TABLE       }
396                          1411,
397                          4250,  {                                 }
398     {  3294  }              0,  {                    Call        }
399                          3327,  {                    Factor      }
400                          4250,  {                                 }
401     {  3297  }              6,  {                    Emit        }
402                            92,  {                    aMultiply   }
403     {  3299  }              3,  {                    Merge       }
404                          3324,  {                                 }
405     {  3301  }              0,  {                    Call        }
406                          3327,  {                    Factor      }
407                          4250,  {                                 }
408     {  3304  }              6,  {                    Emit        }
409                            37,  {                    aDiv        }
410     {  3306  }              3,  {                    Merge       }
411                          3324,  {                                 }
412     {  3308  }              0,  {                    Call        }
413                          3327,  {                    Factor      }
414                          4250,  {                                 }
415     {  3311  }              6,  {                    Emit        }
416                            89,  {                    aMod        }
417     {  3313  }              3,  {                    Merge       }
418                          3324,  {                                 }
419     {  3315  }              3,  {                    TABLE BEGINS }
420                            22,  {                    tAsterisk   }
421                          3294,
422                            62,  {                    tDiv        }
423                          3301,
424                            83,  {                    tMod        }
425                          3308,
426     {  3322  }              3,  {                    Merge       }
427                          3326,  {                                 }
428     {  3324  }              2,  {                    Repeat      }
429                          3290,  {                                 }
430     {  3326  }              1,  {                    Return      }
431     {  3327  }              5,  { Factor             InputChoice }
432                          3379,  {                    TABLE       }
```

```
433                          1426,
434                          4978,  {                          }
435     {  3331  }             0,   {  Call                    }
436                          3407,  {  Variable                }
437                          3570,  {                          }
438     {  3334  }             3,   {  Merge                   }
439                          3406,  {                          }
440     {  3336  }             6,   {  Emit                    }
441                            1,   {  aNumber                 }
442     {  3338  }            11,   {  oEmitNumber             }
443     {  3339  }             3,   {  Merge                   }
444                          3406,  {                          }
445     {  3341  }             6,   {  Emit                    }
446                            4,   {  aChar                   }
447     {  3343  }            10,   {  oEmitChar               }
448     {  3344  }             3,   {  Merge                   }
449                          3406,  {                          }
450     {  3346  }             6,   {  Emit                    }
451                            5,   {  aMDChar                 }
452     {  3348  }            10,   {  oEmitChar               }
453     {  3349  }             3,   {  Merge                   }
454                          3406,  {                          }
455     {  3351  }             6,   {  Emit                    }
456                            2,   {  aString                 }
457     {  3353  }            12,   {  oEmitString             }
458     {  3354  }             3,   {  Merge                   }
459                          3406,  {                          }
460     {  3356  }             6,   {  Emit                    }
461                            3,   {  aMDString               }
462     {  3358  }            12,   {  oEmitString             }
463     {  3359  }             3,   {  Merge                   }
464                          3406,  {                          }
465     {  3361  }             0,   {  Call                    }
466                          3006,  {  SubExpression           }
467                          3978,  {                          }
468     {  3364  }             4,   {  Input                   }
469                           30,   {  tRightParen             }
470                          1440,
471                          3570,  {                          }
472     {  3368  }             6,   {  Emit                    }
473                          101,   {  aParens                 }
474     {  3370  }             3,   {  Merge                   }
475                          3406,  {                          }
476     {  3372  }             0,   {  Call                    }
477                          3327,  {  Factor                  }
478                          3570,  {                          }
479     {  3375  }             6,   {  Emit                    }
480                           88,   {  aMinus                  }
481     {  3377  }             3,   {  Merge                   }
482                          3406,  {                          }
483     {  3379  }            12,   {  TABLE BEGINS            }
484                            0,   {  tIdent                  }
```

```
                                3331,
                                  1,  {                    tNumber          }
                                3336,
                                  2,  {                    tOctalNumber     }
                                3336,
                                  3,  {                    tHexNumber       }
                                3336,
                                  7,  {                    tChar            }
                                3341,
                                  8,  {                    tExChar          }
                                3341,
                                  9,  {                    tMDChar          }
                                3346,
                                  4,  {                    tString          }
                                3351,
                                  5,  {                    tExString        }
                                3351,
                                  6,  {                    tMDString        }
                                3356,
                                 29,  {                    tLeftParen       }
                                3361,
                                 21,  {                    tMinus           }
                                3372,
{  3404  }                        2,  {                    Repeat           }
                                3331,  {                                     }
{  3406  }                        1,  {                    Return           }
{  3407  }                        6,  { Variable           Emit             }
                                  0,  {                    aIdent           }
{  3409  }                        9,  {                    oEmitIdent       }
{  3410  }                        5,  {                    InputChoice      }
                                3438,  {                   TABLE            }
                                1450,
                                4986,  {                                     }
{  3414  }                        6,  {                    Emit             }
                                115,  {                    aSubs            }
{  3416  }                        0,  {                    Call             }
                                3450,  {                   Subscripts       }
                                4986,  {                                     }
{  3419  }                        6,  {                    Emit             }
                                 55,  {                    aEndSubs         }
{  3421  }                        3,  {                    Merge            }
                                3447,  {                                     }
{  3423  }                        6,  {                    Emit             }
                                 63,  {                    aField           }
{  3425  }                        4,  {                    Input            }
                                  0,  {                    tIdent           }
                                1454,
                                4986,  {                                     }
{  3429  }                        6,  {                    Emit             }
                                  0,  {                    aIdent           }
{  3431  }                        9,  {                    oEmitIdent       }
{  3432  }                        3,  {                    Merge            }
```

```
537                          3447,  {                                }
538      {  3434  }            6,  {  Emit                           }
539                           105,  {  aPointer                      }
540      {  3436  }            3,  {  Merge                          }
541                          3447,  {                                }
542      {  3438  }            3,  {  TABLE BEGINS                   }
543                            29,  {  tLeftParen                    }
544                          3414,
545                            31,  {  tPeriod                       }
546                          3423,
547                            37,  {  tUpArrow                      }
548                          3434,
549      {  3445  }            3,  {  Merge                          }
550                          3449,  {                                }
551      {  3447  }            2,  {  Repeat                         }
552                          3410,  {                                }
553      {  3449  }            1,  {  Return                         }
554      {  3450  }            5,  {  Subscripts   InputChoice       }
555                          3474,  {  TABLE                         }
556                          1466,
557                          4994,  {                                }
558      {  3454  }            3,  {  Merge                          }
559                          3521,  {                                }
560      {  3456  }            3,  {  Merge                          }
561                          3502,  {                                }
562      {  3458  }            6,  {  Emit                           }
563                           100,  {  aParameter                    }
564      {  3460  }            3,  {  Merge                          }
565                          3502,  {                                }
566      {  3462  }            6,  {  Emit                           }
567                            16,  {  aAny                          }
568      {  3464  }            3,  {  Merge                          }
569                          3502,  {                                }
570      {  3466  }            6,  {  Emit                           }
571                           120,  {  aUnknown                      }
572      {  3468  }            3,  {  Merge                          }
573                          3502,  {                                }
574      {  3470  }            6,  {  Emit                           }
575                            14,  {  aAll                          }
576      {  3472  }            3,  {  Merge                          }
577                          3502,  {                                }
578      {  3474  }            5,  {  TABLE BEGINS                   }
579                            30,  {  tRightParen                   }
580                          3454,
581                            90,  {  tParameter                    }
582                          3458,
583                            43,  {  tAny                          }
584                          3462,
585                           104,  {  tUnknown                      }
586                          3466,
587                            41,  {  tAll                          }
588                          3470,
```

```
589         {   3485    }       0,  {                       Call            }
590                             3000,  {                    Expression      }
591                             4994,  {                                    }
592         {   3488    }       5,  {                       InputChoice     }
593                             3499,  {                    TABLE           }
594                             1479,
595                             4994,  {                                    }
596         {   3492    }       6,  {                       Emit            }
597                              117,  {                    aTo             }
598         {   3494    }       0,  {                       Call            }
599                             3224,  {                    Sum             }
600                             4994,  {                                    }
601         {   3497    }       3,  {                       Merge           }
602                             3502,  {                                    }
603         {   3499    }       1,  {                       TABLE BEGINS    }
604                              32,  {                     tDoublePeriod   }
605                             3492,
606         {   3502    }       5,  {                       InputChoice     }
607                             3512,  {                    TABLE           }
608                             1485,
609                             5002,  {                                    }
610         {   3506    }       3,  {                       Merge           }
611                             3521,  {                                    }
612         {   3508    }       3,  {                       Merge           }
613                             3519,  {                                    }
614         {   3510    }       3,  {                       Merge           }
615                             3519,  {                                    }
616         {   3512    }       2,  {                       TABLE BEGINS    }
617                              30,  {                     tRightParen     }
618                             3506,
619                              34,  {                     tComma          }
620                             3510,
621         {   3517    }       2,  {                       Repeat          }
622                             3506,  {                                    }
623         {   3519    }       2,  {                       Repeat          }
624                             3450,  {                                    }
625         {   3521    }       1,  {                       Return          }
626         {   3522    }       0,      0,      0,      0,      0,      0,  8194,
627                         { tPervasive    tType       }
628         {   3522    }       0,      0,      0,      0,      0,      0,  8194,
629                         { tPervasive    tType       }
630         {   3530    }       0,      0,      0,      0,      0,      0,     2,
631                         { tPervasive    }
632         {   3538    }       1,      0,     32,      0,      0,      0,  8192,
633                         { tIdent        tSemicolon      tType
634                          }
635         {   3546    }       1,      0,     32,      0,      0,      0,     0,
636                         { tIdent        tSemicolon      }
637         {   3554    }       1,      0,      0,      0,      0,      0,     0,
638                         { tIdent        }
639         {   3562    }       0,      0,     32,      0,      0,      0,     0,
640                         { tSemicolon    }
```

```
641     {   3570    }       0,    0,    0,    0,    0,    0,    0,
642                     {  }
643     {   3578    }       1,    8,    0,    0,    0,    0,    0,
644                     { tIdent        tNewLine        }
645     {   3586    }     723, 32744, 4641, 6925, 11140, 5435, 8222,
646                     { tIdent        tNumber         tString
647                     tMDString       tChar           tMDChar
648                     tNewLine        tPlus           tMinus
649                     tAsterisk       tEqual          tLessThan
650                     tGreaterThan    tLessEqual      tGreaterEqual
651                     tImplies        tLeftParen      tRightParen
652                     tSemicolon      tAbstraction    tAnd
653                     tAssert         tBegin          tBind
654                     tChecked        tCode           tConst
655                     tConverter      tDiv            tExports
656                     tExternal       tFinally        tForward
657                     tFunction       tImports        tIn
658                     tInitially      tInline         tInvariant
659                     tMod            tNot            tOr
660                     tPervasive      tPost           tPre
661                     tProcedure      tType           tVar
662                     tXor            }
663     {   3594    }     723, 32736, 4641, 6925, 11140, 5435, 8222,
664                     { tIdent        tNumber         tString
665                     tMDString       tChar           tMDChar
666                     tPlus           tMinus          tAsterisk
667                     tEqual          tLessThan       tGreaterThan
668                     tLessEqual      tGreaterEqual   tImplies
669                     tLeftParen      tRightParen     tSemicolon
670                     tAbstraction    tAnd            tAssert
671                     tBegin          tBind           tChecked
672                     tCode           tConst          tConverter
673                     tDiv            tExports        tExternal
674                     tFinally        tForward        tFunction
675                     tImports        tIn             tInitially
676                     tInline         tInvariant      tMod
677                     tNot            tOr             tPervasive
678                     tPost           tPre            tProcedure
679                     tType           tVar            tXor
680                     }
681     {   3602    }       0,    0,    0,    0,    0,    1,    0,
682                     { tImports      }
683     {   3610    }       0,    0,    0,    0,  128,    0,    0,
684                     { tExports      }
685     {   3618    }     723, 32736, 4641, 6925, 11012, 5435, 8222,
686                     { tIdent        tNumber         tString
687                     tMDString       tChar           tMDChar
688                     tPlus           tMinus          tAsterisk
689                     tEqual          tLessThan       tGreaterThan
690                     tLessEqual      tGreaterEqual   tImplies
691                     tLeftParen      tRightParen     tSemicolon
692                     tAbstraction    tAnd            tAssert
```

```
693                     tBegin          tBind           tChecked
694                     tCode           tConst          tConverter
695                     tDiv            tExternal       tFinally
696                     tForward        tFunction       tImports
697                     tIn             tInitially      tInline
698                     tInvariant      tMod            tNot
699                     tOr             tPervasive      tPost
700                     tPre            tProcedure      tType
701                     tVar            tXor             }
702     {  3626  }       723, 32736,  4641,  6669, 11012,  5435,  8222,
703                     { tIdent         tNumber         tString
704                     tMDString       tChar           tMDChar
705                     tPlus           tMinus          tAsterisk
706                     tEqual          tLessThan       tGreaterThan
707                     tLessEqual      tGreaterEqual   tImplies
708                     tLeftParen      tRightParen     tSemicolon
709                     tAbstraction    tAnd            tAssert
710                     tBegin          tBind           tCode
711                     tConst          tConverter      tDiv
712                     tExternal       tFinally        tForward
713                     tFunction       tImports        tIn
714                     tInitially      tInline         tInvariant
715                     tMod            tNot            tOr
716                     tPervasive      tPost           tPre
717                     tProcedure      tType           tVar
718                     tXor             }
719     {  3634  }          0,     0,   32,  6153,  8192,    16,  8210,
720                     { tSemicolon     tAssert         tBind
721                     tConst          tConverter      tFunction
722                     tInline         tPervasive      tProcedure
723                     tType           tVar             }
724     {  3642  }       723, 32736,  4641,   516, 11012,  5419,    12,
725                     { tIdent         tNumber         tString
726                     tMDString       tChar           tMDChar
727                     tPlus           tMinus          tAsterisk
728                     tEqual          tLessThan       tGreaterThan
729                     tLessEqual      tGreaterEqual   tImplies
730                     tLeftParen      tRightParen     tSemicolon
731                     tAbstraction    tAnd            tBegin
732                     tCode           tDiv            tExternal
733                     tFinally        tForward        tFunction
734                     tImports        tIn             tInitially
735                     tInvariant      tMod            tNot
736                     tOr             tPost           tPre
737                     tXor             }
738     {  3650  }          0,     0,    0,   516,  2304,     1,    12,
739                     { tBegin         tCode           tExternal
740                     tForward        tImports        tPost
741                     tPre             }
742     {  3658  }       723, 32736,  4641,   516, 11012,  5411,    12,
743                     { tIdent         tNumber         tString
744                     tMDString       tChar           tMDChar
```

```
745                     tPlus           tMinus          tAsterisk
746                     tEqual          tLessThan       tGreaterThan
747                     tLessEqual      tGreaterEqual   tImplies
748                     tLeftParen      tRightParen     tSemicolon
749                     tAbstraction    tAnd            tBegin
750                     tCode           tDiv            tExternal
751                     tFinally        tForward        tFunction
752                     tImports        tIn             tInvariant
753                     tMod            tNot            tOr
754                     tPost           tPre            tXor
755                      }
756     {  3666  }       723, 32736,  4129,  516,  2820, 5411,   12,
757                     { tIdent        tNumber         tString
758                     tMDString       tChar           tMDChar
759                     tPlus           tMinus          tAsterisk
760                     tEqual          tLessThan       tGreaterThan
761                     tLessEqual      tGreaterEqual   tImplies
762                     tLeftParen      tRightParen     tSemicolon
763                     tAnd            tBegin          tCode
764                     tDiv            tExternal       tFinally
765                     tForward        tImports        tIn
766                     tInvariant      tMod            tNot
767                     tOr             tPost           tPre
768                     tXor             }
769     {  3674  }       723, 32736,  4129,  516,  2820, 5379,   12,
770                     { tIdent        tNumber         tString
771                     tMDString       tChar           tMDChar
772                     tPlus           tMinus          tAsterisk
773                     tEqual          tLessThan       tGreaterThan
774                     tLessEqual      tGreaterEqual   tImplies
775                     tLeftParen      tRightParen     tSemicolon
776                     tAnd            tBegin          tCode
777                     tDiv            tExternal       tFinally
778                     tForward        tImports        tIn
779                     tMod            tNot            tOr
780                     tPost           tPre            tXor
781                      }
782     {  3682  }       723, 32736,  4096,    0,     4, 5378,    0,
783                     { tIdent        tNumber         tString
784                     tMDString       tChar           tMDChar
785                     tPlus           tMinus          tAsterisk
786                     tEqual          tLessThan       tGreaterThan
787                     tLessEqual      tGreaterEqual   tImplies
788                     tLeftParen      tAnd            tDiv
789                     tIn             tMod            tNot
790                     tOr             tXor             }
791     {  3690  }         0,    0,    33,  516,  2816,    1,   12,
792                     { tRightParen   tSemicolon      tBegin
793                     tCode           tExternal       tFinally
794                     tForward        tImports        tPost
795                     tPre             }
796     {  3698  }         0,    0,    32,  516,  2816,    1,   12,
```

```
797                     { tSemicolon    tBegin      tCode
798                     tExternal       tFinally    tForward
799                     tImports        tPost       tPre
800                      }
801     {  3706  }        1, 16384,     49, 2048,    0,    1, 2082,
802                     { tIdent        tLeftParen  tRightParen
803                     tComma          tSemicolon  tConst
804                     tImports        tPervasive  tReadonly
805                     tThus           tVar         }
806     {  3714  }        1,    0,      0, 2048,    0,    0,   34,
807                     { tIdent        tConst      tPervasive
808                     tReadonly       tVar         }
809     {  3722  }        1,    0,      0, 2048,    0,    0,   32,
810                     { tIdent        tConst      tReadonly
811                     tVar             }
812     {  3730  }        0,    0,      0, 2048,    0,    0,   32,
813                     { tConst        tReadonly   tVar
814                      }
815     {  3738  }        0, 16384,     32,    0,  128,    0,    0,
816                     { tLeftParen    tSemicolon  tExports
817                      }
818     {  3746  }        0, 16384,      0,    0,    0,    0,    0,
819                     { tLeftParen     }
820     {  3754  }        1,  256,    153, 2048,    0,    0,   32,
821                     { tIdent        tEqual      tRightParen
822                     tAssign         tComma      tUpArrow
823                     tConst          tReadonly   tVar
824                      }
825     {  3762  }        1,  256,    136, 2048,    0,    0,   32,
826                     { tIdent        tEqual      tAssign
827                     tUpArrow        tConst      tReadonly
828                     tVar             }
829     {  3770  }        1,  256,    140, 2048,    0,    0,   32,
830                     { tIdent        tEqual      tDoublePeriod
831                     tAssign         tUpArrow    tConst
832                     tReadonly       tVar        tWith
833                      }
834     {  3778  }        1,    0,      4,    0,    0,    0,    0,
835                     { tIdent        tDoublePeriod tWith
836                      }
837     {  3786  }        0,    0,      0,    0,    0,    0,    0,
838                     { tWith          }
839     {  3794  }        1,    0,      0,    0,    0,    0,    0,
840                     { tIdent        tWith        }
841     {  3802  }        0, 16384,     0,    0,    0,    0,    0,
842                     { tLeftParen    tWith        }
843     {  3810  }        1, 16384,    32, 6153, 8192,   16, 8242,
844                     { tIdent        tLeftParen  tSemicolon
845                     tAssert         tBind       tConst
846                     tConverter      tFunction   tInline
847                     tPervasive      tProcedure  tReadonly
848                     tType           tVar         }
```

```
849     {  3818  }     1, 16384,    0,    0,    0,    0,   32,
850                  { tIdent       tLeftParen    tReadonly
851                  tVar          }
852     {  3826  }     0,    0,     0,    0,  8192,    0,   16,
853                  { tFunction    tProcedure    }
854     {  3834  }   723, 32736, 20701,  9346,    4, 22402,  576,
855                  { tIdent       tNumber       tString
856                  tMDString      tChar         tMDChar
857                  tPlus          tMinus        tAsterisk
858                  tEqual         tLessThan     tGreaterThan
859                  tLessEqual     tGreaterEqual tImplies
860                  tLeftParen     tRightParen   tDoublePeriod
861                  tAssign        tComma        tColon
862                  tUpArrow       tAnd          tArray
863                  tAt            tCheckable    tCollection
864                  tCounted       tDiv          tIn
865                  tMachine       tMod          tModule
866                  tNot           tOr           tPacked
867                  tRecord        tSet          tXor
868                   }
869     {  3842  }   723, 32736, 20685,  9344,    4, 22402,  576,
870                  { tIdent       tNumber       tString
871                  tMDString      tChar         tMDChar
872                  tPlus          tMinus        tAsterisk
873                  tEqual         tLessThan     tGreaterThan
874                  tLessEqual     tGreaterEqual tImplies
875                  tLeftParen     tRightParen   tDoublePeriod
876                  tAssign        tColon        tUpArrow
877                  tAnd           tArray        tCheckable
878                  tCollection    tCounted      tDiv
879                  tIn            tMachine      tMod
880                  tModule        tNot          tOr
881                  tPacked        tRecord       tSet
882                  tXor           }
883     {  3850  }   723, 32736, 20684,  9344,    4, 22402,  576,
884                  { tIdent       tNumber       tString
885                  tMDString      tChar         tMDChar
886                  tPlus          tMinus        tAsterisk
887                  tEqual         tLessThan     tGreaterThan
888                  tLessEqual     tGreaterEqual tImplies
889                  tLeftParen     tDoublePeriod tAssign
890                  tColon         tUpArrow      tAnd
891                  tArray         tCheckable    tCollection
892                  tCounted       tDiv          tIn
893                  tMachine       tMod          tModule
894                  tNot           tOr           tPacked
895                  tRecord        tSet          tXor
896                   }
897     {  3858  }   723, 32736, 20700,  9344,    4, 22402,  576,
898                  { tIdent       tNumber       tString
899                  tMDString      tChar         tMDChar
900                  tPlus          tMinus        tAsterisk
```

```
901                     tEqual          tLessThan       tGreaterThan
902                     tLessEqual      tGreaterEqual   tImplies
903                     tLeftParen      tDoublePeriod   tAssign
904                     tComma          tColon          tUpArrow
905                     tAnd            tArray          tCheckable
906                     tCollection     tCounted        tDiv
907                     tIn             tMachine        tMod
908                     tModule         tNot            tOr
909                     tPacked         tRecord         tSet
910                     tXor             }
911     {   3866    }     723, 32736, 20620,  9344,     4, 22402,   576,
912                     { tIdent        tNumber         tString
913                     tMDString       tChar           tMDChar
914                     tPlus           tMinus          tAsterisk
915                     tEqual          tLessThan       tGreaterThan
916                     tLessEqual      tGreaterEqual   tImplies
917                     tLeftParen      tDoublePeriod   tAssign
918                     tUpArrow        tAnd            tArray
919                     tCheckable      tCollection     tCounted
920                     tDiv            tIn             tMachine
921                     tMod            tModule         tNot
922                     tOr             tPacked         tRecord
923                     tSet            tXor             }
924     {   3874    }     723, 16608, 16516,  9344,     4, 17280,   576,
925                     { tIdent        tNumber         tString
926                     tMDString       tChar           tMDChar
927                     tPlus           tMinus          tAsterisk
928                     tLeftParen      tDoublePeriod   tUpArrow
929                     tArray          tCheckable      tCollection
930                     tCounted        tDiv            tMachine
931                     tMod            tModule         tPacked
932                     tRecord         tSet            tXor
933                      }
934     {   3882    }     723, 32736,  4104,     0,     4,  5378,     0,
935                     { tIdent        tNumber         tString
936                     tMDString       tChar           tMDChar
937                     tPlus           tMinus          tAsterisk
938                     tEqual          tLessThan       tGreaterThan
939                     tLessEqual      tGreaterEqual   tImplies
940                     tLeftParen      tAssign         tAnd
941                     tDiv            tIn             tMod
942                     tNot            tOr             tXor
943                      }
944     {   3890    }        1, 16384,    17,     0,     0,     0,    32,
945                     { tIdent        tLeftParen      tRightParen
946                     tComma          tReadonly       tVar
947                      }
948     {   3898    }        1,     0,     0,     0,     0,     0,    32,
949                     { tIdent        tReadonly       tVar
950                      }
951     {   3906    }        1,     0,    17,     0,     0,     0,    32,
952                     { tIdent        tRightParen     tComma
```

```
953                                 tReadonly       tVar            }
954         {   3914    }           0,    0,        0,    0,        0,    0,    32,
955                                 { tReadonly     tVar            }
956         {   3922    }           1, 16384,       130,    0,      0,    0,  4096,
957                                 { tIdent        tLeftParen      tPeriod
958                                 tUpArrow        tTo             }
959         {   3930    }           1, 16384,       130,    0,      0,    0,    0,
960                                 { tIdent        tLeftParen      tPeriod
961                                 tUpArrow        }
962         {   3938    }           0, 16384,       130,    0,      0,    0,    0,
963                                 { tLeftParen    tPeriod         tUpArrow
964                                 }
965         {   3946    }           723, 32736,  20701,  9344,      4, 22402,   576,
966                                 { tIdent        tNumber         tString
967                                 tMDString       tChar           tMDChar
968                                 tPlus           tMinus          tAsterisk
969                                 tEqual          tLessThan       tGreaterThan
970                                 tLessEqual      tGreaterEqual   tImplies
971                                 tLeftParen      tRightParen     tDoublePeriod
972                                 tAssign         tComma          tColon
973                                 tUpArrow        tAnd            tArray
974                                 tCheckable      tCollection     tCounted
975                                 tDiv            tIn             tMachine
976                                 tMod            tModule         tNot
977                                 tOr             tPacked         tRecord
978                                 tSet            tXor            }
979         {   3954    }           723, 32736,  4121,    0,        4,  5378,     0,
980                                 { tIdent        tNumber         tString
981                                 tMDString       tChar           tMDChar
982                                 tPlus           tMinus          tAsterisk
983                                 tEqual          tLessThan       tGreaterThan
984                                 tLessEqual      tGreaterEqual   tImplies
985                                 tLeftParen      tRightParen     tAssign
986                                 tComma          tAnd            tDiv
987                                 tIn             tMod            tNot
988                                 tOr             tXor            }
989         {   3962    }           723, 32736,  4113,    0,        4,  5378,     0,
990                                 { tIdent        tNumber         tString
991                                 tMDString       tChar           tMDChar
992                                 tPlus           tMinus          tAsterisk
993                                 tEqual          tLessThan       tGreaterThan
994                                 tLessEqual      tGreaterEqual   tImplies
995                                 tLeftParen      tRightParen     tComma
996                                 tAnd            tDiv            tIn
997                                 tMod            tNot            tOr
998                                 tXor            }
999         {   3970    }           723, 32736,  4112,    0,        4,  5378,     0,
000                                 { tIdent        tNumber         tString
001                                 tMDString       tChar           tMDChar
002                                 tPlus           tMinus          tAsterisk
003                                 tEqual          tLessThan       tGreaterThan
004                                 tLessEqual      tGreaterEqual   tImplies
```

```
005                          tLeftParen     tComma         tAnd
006                          tDiv           tIn            tMod
007                          tNot           tOr            tXor
008                           }
009       {  3978  }           0,    0,     1,    0,       0,    0,    0,
010                          { tRightParen   }
011       {  3986  }           0,    0,    16,    0,       0,    0,    0,
012                          { tComma        }
013       {  3994  }           1, 16384,     1,    0,    8192,    0,  272,
014                          { tIdent        tLeftParen     tRightParen
015                          tFunction      tProcedure     tReturns
016                           }
017       {  4002  }           1,    0,     1,    0,    8192,    0,  272,
018                          { tIdent        tRightParen    tFunction
019                          tProcedure     tReturns        }
020       {  4010  }           1,    0,     0,    0,       0,    0,  256,
021                          { tIdent        tReturns        }
022       {  4018  }         723, 32744, 20661, 11392,  2052, 22402,  586,
023                          { tIdent        tNumber        tString
024                          tMDString      tChar          tMDChar
025                          tNewLine       tPlus          tMinus
026                          tAsterisk      tEqual         tLessThan
027                          tGreaterThan   tLessEqual     tGreaterEqual
028                          tImplies       tLeftParen     tRightParen
029                          tDoublePeriod  tComma          tSemicolon
030                          tUpArrow       tAnd           tArray
031                          tCheckable     tCollection    tConst
032                          tCounted       tDiv           tForward
033                          tIn            tMachine       tMod
034                          tModule        tNot           tOr
035                          tPacked        tPervasive     tPre
036                          tRecord        tSet           tXor
037                           }
038       {  4026  }           1,    0,     0, 2048,       0,    0,    2,
039                          { tIdent        tConst         tPervasive
040                           }
041       {  4034  }         723, 32744, 20645,  9344,  2052, 22402,  584,
042                          { tIdent        tNumber        tString
043                          tMDString      tChar          tMDChar
044                          tNewLine       tPlus          tMinus
045                          tAsterisk      tEqual         tLessThan
046                          tGreaterThan   tLessEqual     tGreaterEqual
047                          tImplies       tLeftParen     tRightParen
048                          tDoublePeriod  tSemicolon     tUpArrow
049                          tAnd           tArray         tCheckable
050                          tCollection    tCounted       tDiv
051                          tForward       tIn            tMachine
052                          tMod           tModule        tNot
053                          tOr            tPacked        tPre
054                          tRecord        tSet           tXor
055                           }
056       {  4042  }         723, 32736, 20645,  9344,  2052, 22402,  584,
```

```
057                     { tIdent        tNumber         tString
058                     tMDString       tChar           tMDChar
059                     tPlus           tMinus          tAsterisk
060                     tEqual          tLessThan       tGreaterThan
061                     tLessEqual      tGreaterEqual   tImplies
062                     tLeftParen      tRightParen     tDoublePeriod
063                     tSemicolon      tUpArrow        tAnd
064                     tArray          tCheckable      tCollection
065                     tCounted        tDiv            tForward
066                     tIn             tMachine        tMod
067                     tModule         tNot            tOr
068                     tPacked         tPre            tRecord
069                     tSet            tXor             }
070     {  4050  }        723, 32736, 20645,  9344,  2052, 22402,   576,
071                     { tIdent        tNumber         tString
072                     tMDString       tChar           tMDChar
073                     tPlus           tMinus          tAsterisk
074                     tEqual          tLessThan       tGreaterThan
075                     tLessEqual      tGreaterEqual   tImplies
076                     tLeftParen      tRightParen     tDoublePeriod
077                     tSemicolon      tUpArrow        tAnd
078                     tArray          tCheckable      tCollection
079                     tCounted        tDiv            tForward
080                     tIn             tMachine        tMod
081                     tModule         tNot            tOr
082                     tPacked         tRecord         tSet
083                     tXor             }
084     {  4058  }        723, 16608, 16549,  9344,  2052, 17280,   576,
085                     { tIdent        tNumber         tString
086                     tMDString       tChar           tMDChar
087                     tPlus           tMinus          tAsterisk
088                     tLeftParen      tRightParen     tDoublePeriod
089                     tSemicolon      tUpArrow        tArray
090                     tCheckable      tCollection     tCounted
091                     tDiv            tForward        tMachine
092                     tMod            tModule         tPacked
093                     tRecord         tSet            tXor
094                      }
095     {  4066  }        723, 16608, 16548,  9344,  2052, 17280,   576,
096                     { tIdent        tNumber         tString
097                     tMDString       tChar           tMDChar
098                     tPlus           tMinus          tAsterisk
099                     tLeftParen      tDoublePeriod   tSemicolon
100                     tUpArrow        tArray          tCheckable
101                     tCollection     tCounted        tDiv
102                     tForward        tMachine        tMod
103                     tModule         tPacked         tRecord
104                     tSet            tXor             }
105     {  4074  }        723, 16608, 16516,  9344,  2052, 17280,   576,
106                     { tIdent        tNumber         tString
107                     tMDString       tChar           tMDChar
108                     tPlus           tMinus          tAsterisk
```

```
109                     tLeftParen      tDoublePeriod   tUpArrow
110                     tArray          tCheckable      tCollection
111                     tCounted        tDiv            tForward
112                     tMachine        tMod            tModule
113                     tPacked         tRecord         tSet
114                     tXor             }
115     {  4082  }       723, 16608,     84,  2048,      4,   256,     0,
116                     { tIdent        tNumber         tString
117                     tMDString       tChar           tMDChar
118                     tPlus           tMinus          tAsterisk
119                     tLeftParen      tDoublePeriod   tComma
120                     tColon          tConst          tDiv
121                     tMod            tXor             }
122     {  4090  }       723, 16608,     84,     0,      4,   256,     0,
123                     { tIdent        tNumber         tString
124                     tMDString       tChar           tMDChar
125                     tPlus           tMinus          tAsterisk
126                     tLeftParen      tDoublePeriod   tComma
127                     tColon          tDiv            tMod
128                     tXor             }
129     {  4098  }       723, 16608,     4,     0,       4,   256,     0,
130                     { tIdent        tNumber         tString
131                     tMDString       tChar           tMDChar
132                     tPlus           tMinus          tAsterisk
133                     tLeftParen      tDoublePeriod   tDiv
134                     tMod            tXor             }
135     {  4106  }          1,     0,     0,     0,  8192,     0,    16,
136                     { tIdent        tFunction       tProcedure
137                      }
138     {  4114  }       723, 32744, 21670, 11457,     38, 24450,   578,
139                     { tIdent        tNumber         tString
140                     tMDString       tChar           tMDChar
141                     tNewLine        tPlus           tMinus
142                     tAsterisk       tEqual          tLessThan
143                     tGreaterThan    tLessEqual      tGreaterEqual
144                     tImplies        tLeftParen      tPeriod
145                     tDoublePeriod   tSemicolon      tUpArrow
146                     tAligned        tAnd            tArray
147                     tAssert         tCase           tCheckable
148                     tCollection     tConst          tCounted
149                     tDependent      tDiv            tEnd
150                     tIn             tMachine        tMod
151                     tModule         tNot            tOf
152                     tOr             tPacked         tPervasive
153                     tRecord         tSet            tVar
154                     tXor             }
155     {  4122  }       723, 16608, 16516,  9344,      4, 19328,   576,
156                     { tIdent        tNumber         tString
157                     tMDString       tChar           tMDChar
158                     tPlus           tMinus          tAsterisk
159                     tLeftParen      tDoublePeriod   tUpArrow
160                     tArray          tCheckable      tCollection
```

```
161                     tCounted        tDiv            tMachine
162                     tMod            tModule         tOf
163                     tPacked         tRecord         tSet
164                     tXor             }
165     {  4130  }        0,    0,      32, 2113,        0,    0,     2,
166                     { tSemicolon     tAssert         tCase
167                     tConst          tPervasive      tVar
168                      }
169     {  4138  }        0,    0,       0,    0,       32,    0,     0,
170                     { tEnd           }
171     {  4146  }      723, 32744,    5152, 2049,       36, 5890,    66,
172                     { tIdent         tNumber          tString
173                     tMDString       tChar           tMDChar
174                     tNewLine        tPlus           tMinus
175                     tAsterisk       tEqual          tLessThan
176                     tGreaterThan    tLessEqual      tGreaterEqual
177                     tImplies        tLeftParen      tSemicolon
178                     tAligned        tAnd            tAssert
179                     tConst          tDiv            tEnd
180                     tIn             tMod            tModule
181                     tNot            tOr             tPervasive
182                     tRecord         tVar            tXor
183                      }
184     {  4154  }      723, 32736,    5152, 2049,       36, 5378,    2,
185                     { tIdent         tNumber          tString
186                     tMDString       tChar           tMDChar
187                     tPlus           tMinus          tAsterisk
188                     tEqual          tLessThan       tGreaterThan
189                     tLessEqual      tGreaterEqual   tImplies
190                     tLeftParen      tSemicolon      tAligned
191                     tAnd            tAssert         tConst
192                     tDiv            tEnd            tIn
193                     tMod            tNot            tOr
194                     tPervasive      tVar            tXor
195                      }
196     {  4162  }      723, 32736,    4128, 2049,       36, 5378,    2,
197                     { tIdent         tNumber          tString
198                     tMDString       tChar           tMDChar
199                     tPlus           tMinus          tAsterisk
200                     tEqual          tLessThan       tGreaterThan
201                     tLessEqual      tGreaterEqual   tImplies
202                     tLeftParen      tSemicolon      tAnd
203                     tAssert         tConst          tDiv
204                     tEnd            tIn             tMod
205                     tNot            tOr             tPervasive
206                     tVar            tXor             }
207     {  4170  }        0,    0,      32, 2049,        32,    0,     2,
208                     { tSemicolon     tAssert          tConst
209                     tEnd            tPervasive      tVar
210                      }
211     {  4178  }        0,    0,      32, 2049,        0,    0,     2,
212                     { tSemicolon     tAssert          tConst
```

```
213                     tPervasive      tVar            }
214     {  4186  }        0,    0,      0,    0,        0, 2048,    0,
215                     { tOf           }
216     {  4194  }      723, 32736,   4096, 1024,       4, 7426,    0,
217                     { tIdent        tNumber         tString
218                     tMDString       tChar           tMDChar
219                     tPlus           tMinus          tAsterisk
220                     tEqual          tLessThan       tGreaterThan
221                     tLessEqual      tGreaterEqual   tImplies
222                     tLeftParen      tAnd            tCollection
223                     tDiv            tIn             tMod
224                     tNot            tOf             tOr
225                     tXor            }
226     {  4202  }        0,    0,      0, 1024,        0, 2048,    0,
227                     { tCollection   tOf             }
228     {  4210  }      723, 16616,  16548, 11457,     38, 19328,  578,
229                     { tIdent        tNumber         tString
230                     tMDString       tChar           tMDChar
231                     tNewLine        tPlus           tMinus
232                     tAsterisk       tLeftParen      tDoublePeriod
233                     tSemicolon      tUpArrow        tArray
234                     tAssert         tCase           tCheckable
235                     tCollection     tConst          tCounted
236                     tDependent      tDiv            tEnd
237                     tMachine        tMod            tModule
238                     tOf             tPacked         tPervasive
239                     tRecord         tSet            tVar
240                     tXor            }
241     {  4218  }        1,    8,      0,    0,       32,  512,    0,
242                     { tIdent        tNewLine        tEnd
243                     tModule         }
244     {  4226  }        1,    8,      0,    0,       32,    0,    0,
245                     { tIdent        tNewLine        tEnd
246                     }
247     {  4234  }        1,    0,     17,    0,        0,    0,    0,
248                     { tIdent        tRightParen     tComma
249                     }
250     {  4242  }      723, 16608,    134,    0,       4,  256,    0,
251                     { tIdent        tNumber         tString
252                     tMDString       tChar           tMDChar
253                     tPlus           tMinus          tAsterisk
254                     tLeftParen      tPeriod         tDoublePeriod
255                     tUpArrow        tDiv            tMod
256                     tXor            }
257     {  4250  }      723, 16576,     0,    0,        4,  256,    0,
258                     { tIdent        tNumber         tString
259                     tMDString       tChar           tMDChar
260                     tMinus          tAsterisk       tLeftParen
261                     tDiv            tMod            }
262     {  4258  }        0,  224,      0,    0,        4,  256,    0,
263                     { tPlus         tMinus          tAsterisk
264                     tDiv            tMod            tXor
```

```
                              }
      {  4266  }               723, 16448,    0,    0,     0,    0,      0,
                              { tIdent      tNumber      tString
                              tMDString    tChar         tMDChar
                              tMinus       tLeftParen    }
      {  4274  }               723, 16608,    0,    0,     4,  256,      0,
                              { tIdent      tNumber      tString
                              tMDString    tChar         tMDChar
                              tPlus        tMinus        tAsterisk
                              tLeftParen   tDiv          tMod
                              tXor          }
      {  4282  }               723, 32744, 4404, 2113,    37, 15618,     2,
                              { tIdent      tNumber      tString
                              tMDString    tChar         tMDChar
                              tNewLine     tPlus         tMinus
                              tAsterisk    tEqual        tLessThan
                              tGreaterThan tLessEqual    tGreaterEqual
                              tImplies     tLeftParen    tDoublePeriod
                              tComma       tSemicolon    tLabel
                              tAnd         tAssert       tCase
                              tConst       tDefault      tDiv
                              tEnd         tIn           tMod
                              tNot         tOf           tOr
                              tOtherwise   tPervasive    tVar
                              tXor          }
      {  4290  }               723, 32744, 4404, 2113,    36, 15618,     2,
                              { tIdent      tNumber      tString
                              tMDString    tChar         tMDChar
                              tNewLine     tPlus         tMinus
                              tAsterisk    tEqual        tLessThan
                              tGreaterThan tLessEqual    tGreaterEqual
                              tImplies     tLeftParen    tDoublePeriod
                              tComma       tSemicolon    tLabel
                              tAnd         tAssert       tCase
                              tConst       tDiv          tEnd
                              tIn          tMod          tNot
                              tOf          tOr           tOtherwise
                              tPervasive   tVar          tXor
                              }
      {  4298  }               723, 32744, 4404, 2113,    36, 13570,     2,
                              { tIdent      tNumber      tString
                              tMDString    tChar         tMDChar
                              tNewLine     tPlus         tMinus
                              tAsterisk    tEqual        tLessThan
                              tGreaterThan tLessEqual    tGreaterEqual
                              tImplies     tLeftParen    tDoublePeriod
                              tComma       tSemicolon    tLabel
                              tAnd         tAssert       tCase
                              tConst       tDiv          tEnd
                              tIn          tMod          tNot
                              tOr          tOtherwise    tPervasive
                              tVar         tXor           }
```

```
317     {  4306  }      0,    8,   32, 2113,   32,    0,    2,
318                   { tNewLine     tSemicolon    tAssert
319                   tCase          tConst        tEnd
320                   tPervasive     tVar           }
321     {  4314  }      0,    0,   32, 2113,   32,    0,    2,
322                   { tSemicolon   tAssert       tCase
323                   tConst         tEnd          tPervasive
324                   tVar           }
325     {  4322  }      0,    0,   32,   64,   32,    0,    0,
326                   { tSemicolon   tCase         tEnd
327                    }
328     {  4330  }      0,    0,   32,   64,    0,    0,    0,
329                   { tSemicolon   tCase          }
330     {  4338  }    723, 32736, 4100,    0,    4, 5378,    0,
331                   { tIdent       tNumber       tString
332                   tMDString      tChar         tMDChar
333                   tPlus          tMinus        tAsterisk
334                   tEqual         tLessThan     tGreaterThan
335                   tLessEqual     tGreaterEqual tImplies
336                   tLeftParen     tDoublePeriod tAnd
337                   tDiv           tIn           tMod
338                   tNot           tOr           tXor
339                    }
340     {  4346  }      1, 16384,   32, 2049,    0,    0,    2,
341                   { tIdent       tLeftParen    tSemicolon
342                   tAssert        tConst        tPervasive
343                   tVar           }
344     {  4354  }      1,    0,    0,    0,    0,    0,   64,
345                   { tIdent       tRecord        }
346     {  4362  }      1,    0,    0,    0,    0,  512,    0,
347                   { tIdent       tModule        }
348     {  4370  }    723, 32736, 20717, 11411,    4, 22402,  578,
349                   { tIdent       tNumber       tString
350                   tMDString      tChar         tMDChar
351                   tPlus          tMinus        tAsterisk
352                   tEqual         tLessThan     tGreaterThan
353                   tLessEqual     tGreaterEqual tImplies
354                   tLeftParen     tRightParen   tDoublePeriod
355                   tAssign        tSemicolon    tColon
356                   tUpArrow       tAnd          tArray
357                   tAssert        tAt           tBits
358                   tCheckable     tCollection   tConst
359                   tCounted       tDiv          tIn
360                   tMachine       tMod          tModule
361                   tNot           tOr           tPacked
362                   tPervasive     tRecord       tSet
363                   tVar           tXor           }
364     {  4378  }    723, 16608, 16518,  9344,    4, 17282,  576,
365                   { tIdent       tNumber       tString
366                   tMDString      tChar         tMDChar
367                   tPlus          tMinus        tAsterisk
368                   tLeftParen     tPeriod       tDoublePeriod
```

```
369                                 tUpArrow        tArray          tCheckable
370                                 tCollection     tCounted        tDiv
371                                 tIn             tMachine        tMod
372                                 tModule         tPacked         tRecord
373                                 tSet            tXor             }
374         {   4386    }             1, 16384,   130,    0,       0,    2,     0,
375                                 { tIdent        tLeftParen      tPeriod
376                                 tUpArrow        tIn             }
377         {   4394    }             0, 16648,     0,  516,    2304,    1,    12,
378                                 { tNewLine      tEqual          tLeftParen
379                                 tBegin          tCode           tExternal
380                                 tForward        tImports        tPost
381                                 tPre             }
382         {   4402    }             0,  264,      0,  516,    2304,    1,    12,
383                                 { tNewLine      tEqual          tBegin
384                                 tCode           tExternal       tForward
385                                 tImports        tPost           tPre
386                                  }
387         {   4410    }             0,    8,      0,  516,    2304,    1,    12,
388                                 { tNewLine      tBegin          tCode
389                                 tExternal       tForward        tImports
390                                 tPost           tPre             }
391         {   4418    }           723, 16872,  16580, 9860,    2308, 17281,   844,
392                                 { tIdent        tNumber         tString
393                                 tMDString       tChar           tMDChar
394                                 tNewLine        tPlus           tMinus
395                                 tAsterisk       tEqual          tLeftParen
396                                 tDoublePeriod   tColon          tUpArrow
397                                 tArray          tBegin          tCheckable
398                                 tCode           tCollection     tCounted
399                                 tDiv            tExternal       tForward
400                                 tImports        tMachine        tMod
401                                 tModule         tPacked         tPost
402                                 tPre            tRecord         tReturns
403                                 tSet            tXor             }
404         {   4426    }           723, 16872,  16580, 9860,    2308, 17281,   588,
405                                 { tIdent        tNumber         tString
406                                 tMDString       tChar           tMDChar
407                                 tNewLine        tPlus           tMinus
408                                 tAsterisk       tEqual          tLeftParen
409                                 tDoublePeriod   tColon          tUpArrow
410                                 tArray          tBegin          tCheckable
411                                 tCode           tCollection     tCounted
412                                 tDiv            tExternal       tForward
413                                 tImports        tMachine        tMod
414                                 tModule         tPacked         tPost
415                                 tPre            tRecord         tSet
416                                 tXor             }
417         {   4434    }           723, 16872,  16516, 9860,    2308, 17281,   588,
418                                 { tIdent        tNumber         tString
419                                 tMDString       tChar           tMDChar
420                                 tNewLine        tPlus           tMinus
```

```
421                     tAsterisk       tEqual          tLeftParen
422                     tDoublePeriod   tUpArrow        tArray
423                     tBegin          tCheckable      tCode
424                     tCollection     tCounted        tDiv
425                     tExternal       tForward        tImports
426                     tMachine        tMod            tModule
427                     tPacked         tPost           tPre
428                     tRecord         tSet            tXor
429                      }
430     {  4442  }        1, 16384,   17,  2048,    0,    0,   34,
431                     { tIdent        tLeftParen      tRightParen
432                     tComma          tConst          tPervasive
433                     tReadonly       tVar             }
434     {  4450  }        1,    0,    17,  2048,    0,    0,   34,
435                     { tIdent        tRightParen     tComma
436                     tConst          tPervasive      tReadonly
437                     tVar             }
438     {  4458  }      723, 16608, 16596, 11392,    4, 17280,  608,
439                     { tIdent        tNumber         tString
440                     tMDString       tChar           tMDChar
441                     tPlus           tMinus          tAsterisk
442                     tLeftParen      tDoublePeriod   tComma
443                     tColon          tUpArrow        tArray
444                     tCheckable      tCollection     tConst
445                     tCounted        tDiv            tMachine
446                     tMod            tModule         tPacked
447                     tReadonly       tRecord         tSet
448                     tVar            tXor             }
449     {  4466  }      723, 16608, 16596,  9344,    4, 17280,  576,
450                     { tIdent        tNumber         tString
451                     tMDString       tChar           tMDChar
452                     tPlus           tMinus          tAsterisk
453                     tLeftParen      tDoublePeriod   tComma
454                     tColon          tUpArrow        tArray
455                     tCheckable      tCollection     tCounted
456                     tDiv            tMachine        tMod
457                     tModule         tPacked         tRecord
458                     tSet            tXor             }
459     {  4474  }      723, 32736,  4129,   516,  2308,  5378,   12,
460                     { tIdent        tNumber         tString
461                     tMDString       tChar           tMDChar
462                     tPlus           tMinus          tAsterisk
463                     tEqual          tLessThan       tGreaterThan
464                     tLessEqual      tGreaterEqual   tImplies
465                     tLeftParen      tRightParen     tSemicolon
466                     tAnd            tBegin          tCode
467                     tDiv            tExternal       tForward
468                     tIn             tMod            tNot
469                     tOr             tPost           tPre
470                     tXor             }
471     {  4482  }      723, 32736,  4129,   516,  2308,  5378,    4,
472                     { tIdent        tNumber         tString
```

```
473                     tMDString       tChar           tMDChar
474                     tPlus           tMinus          tAsterisk
475                     tEqual          tLessThan       tGreaterThan
476                     tLessEqual      tGreaterEqual   tImplies
477                     tLeftParen      tRightParen     tSemicolon
478                     tAnd            tBegin          tCode
479                     tDiv            tExternal       tForward
480                     tIn             tMod            tNot
481                     tOr             tPost           tXor
482                     }
483     {   4490    }     723, 32736,   4129,   516,  2308,  5378,     0,
484                     { tIdent          tNumber         tString
485                     tMDString       tChar           tMDChar
486                     tPlus           tMinus          tAsterisk
487                     tEqual          tLessThan       tGreaterThan
488                     tLessEqual      tGreaterEqual   tImplies
489                     tLeftParen      tRightParen     tSemicolon
490                     tAnd            tBegin          tCode
491                     tDiv            tExternal       tForward
492                     tIn             tMod            tNot
493                     tOr             tXor            }
494     {   4498    }       0,     0,    33,   516,  2304,     0,     0,
495                     { tRightParen     tSemicolon      tBegin
496                     tCode           tExternal       tForward
497                     }
498     {   4506    }       0,     0,    32,   516,  2304,     0,     0,
499                     { tSemicolon      tBegin          tCode
500                     tExternal       tForward        }
501     {   4514    }       0,     0,     0,   516,  2304,     0,     0,
502                     { tBegin          tCode           tExternal
503                     tForward        }
504     {   4522    }   16385,     3,    32,  6925, 10528,  1040,  8210,
505                     { tIdent          tCodeStart      tCodeMiddle
506                     tCodeEnd        tSemicolon      tAssert
507                     tBegin          tBind           tChecked
508                     tCode           tConst          tConverter
509                     tEnd            tExternal       tForward
510                     tFunction       tInline         tNot
511                     tPervasive      tProcedure      tType
512                     tVar            }
513     {   4530    }       0,     0,    32,  6409,  8192,  1040,  8210,
514                     { tSemicolon      tAssert         tBind
515                     tChecked        tConst          tConverter
516                     tFunction       tInline         tNot
517                     tPervasive      tProcedure      tType
518                     tVar            }
519     {   4538    }       1,     0,     0,     0,    32,     0,     0,
520                     { tIdent          tEnd            }
521     {   4546    }   16385,     3,     0,     0,    32,     0,     0,
522                     { tIdent          tCodeStart      tCodeMiddle
523                     tCodeEnd        tEnd            }
524     {   4554    }       1,     3,     0,     0,    32,     0,     0,
```

```
525                                 { tIdent          tCodeMiddle    tCodeEnd
526                                 tEnd              }
527         {   4562    }             1,     0,    32,    69, 17472,    64,   128,
528                                 { tIdent          tSemicolon     tAssert
529                                 tBegin          tCase          tExit
530                                 tFor            tIf            tLoop
531                                 tReturn           }
532         {   4570    }           723, 32744,  4266,  6477, 25700,  7506,  9394,
533                                 { tIdent          tNumber        tString
534                                 tMDString       tChar          tMDChar
535                                 tNewLine        tPlus          tMinus
536                                 tAsterisk       tEqual         tLessThan
537                                 tGreaterThan    tLessEqual     tGreaterEqual
538                                 tImplies        tLeftParen     tPeriod
539                                 tAssign         tSemicolon     tUpArrow
540                                 tAnd            tAssert        tBegin
541                                 tBind           tCase          tChecked
542                                 tConst          tConverter     tDiv
543                                 tEnd            tExit          tFor
544                                 tFunction       tIf            tIn
545                                 tInline         tLoop          tMod
546                                 tNot            tOf            tOr
547                                 tPervasive      tProcedure     tReadonly
548                                 tReturn         tThen          tType
549                                 tVar            tWhen          tXor
550                                  }
551         {   4578    }             0, 16384,   138,     0,     0,     0,     0,
552                                 { tLeftParen      tPeriod        tAssign
553                                 tUpArrow          }
554         {   4586    }             0,     0,     0,     0,     0,     0,     0,
555                                 { tWhen           }
556         {   4594    }             0, 16384,     0,     0,     0,     0,     0,
557                                 { tLeftParen      tWhen          }
558         {   4602    }             0,     0,    32,  6409,  8224,  1040,  8210,
559                                 { tSemicolon      tAssert        tBind
560                                 tChecked        tConst         tConverter
561                                 tEnd            tFunction      tInline
562                                 tNot            tPervasive     tProcedure
563                                 tType           tVar            }
564         {   4610    }           723, 32736,  4096,     0,     4,  5378,  1024,
565                                 { tIdent          tNumber        tString
566                                 tMDString       tChar          tMDChar
567                                 tPlus           tMinus         tAsterisk
568                                 tEqual          tLessThan      tGreaterThan
569                                 tLessEqual      tGreaterEqual  tImplies
570                                 tLeftParen      tAnd           tDiv
571                                 tIn             tMod           tNot
572                                 tOr             tThen          tXor
573                                  }
574         {   4618    }             0,     8,     0,     0,     0,     0,     0,
575                                 { tNewLine        }
576         {   4626    }           723, 16608,     0,  2048,     4,  2304,    32,
```

```
577                     { tIdent        tNumber         tString
578                     tMDString       tChar           tMDChar
579                     tPlus           tMinus          tAsterisk
580                     tLeftParen      tConst          tDiv
581                     tMod            tOf             tReadonly
582                     tVar            tXor             }
583     {  4634  }        723, 32736,   4096,    0,       4,  5378,    0,
584                     { tIdent        tNumber         tString
585                     tMDString       tChar           tMDChar
586                     tPlus           tMinus          tAsterisk
587                     tEqual          tLessThan       tGreaterThan
588                     tLessEqual      tGreaterEqual   tImplies
589                     tLeftParen      tAnd            tDiv
590                     tIn             tMod            tNot
591                     tOr             tWhen           tXor
592                      }
593     {  4642  }        723, 32736,   4097,    0,       4,  5378,    0,
594                     { tIdent        tNumber         tString
595                     tMDString       tChar           tMDChar
596                     tPlus           tMinus          tAsterisk
597                     tEqual          tLessThan       tGreaterThan
598                     tLessEqual      tGreaterEqual   tImplies
599                     tLeftParen      tRightParen     tAnd
600                     tDiv            tIn             tMod
601                     tNot            tOr             tWhen
602                     tXor             }
603     {  4650  }        723, 32736,   4097,    0,       4,  5378,    0,
604                     { tIdent        tNumber         tString
605                     tMDString       tChar           tMDChar
606                     tPlus           tMinus          tAsterisk
607                     tEqual          tLessThan       tGreaterThan
608                     tLessEqual      tGreaterEqual   tImplies
609                     tLeftParen      tRightParen     tAnd
610                     tDiv            tIn             tMod
611                     tNot            tOr             tXor
612                      }
613     {  4658  }          1,    0,     32,  6477, 25664,  1104,  8338,
614                     { tIdent        tSemicolon      tAssert
615                     tBegin          tBind           tCase
616                     tChecked        tConst          tConverter
617                     tExit           tFor            tFunction
618                     tIf             tInline         tLoop
619                     tNot            tPervasive      tProcedure
620                     tReturn         tType           tVar
621                      }
622     {  4666  }          1,    0,     32,  6221, 25664,    80,  8338,
623                     { tIdent        tSemicolon      tAssert
624                     tBegin          tBind           tCase
625                     tConst          tConverter      tExit
626                     tFor            tFunction       tIf
627                     tInline         tLoop           tPervasive
628                     tProcedure      tReturn         tType
```

```
629                                 tVar            }
630         {   4674    }       0,      8,    32,  6409, 24632,  1040,  9234,
631                             { tNewLine      tSemicolon      tAssert
632                             tBind           tChecked        tConst
633                             tConverter      tElse           tElseif
634                             tEnd            tFunction       tIf
635                             tInline         tNot            tPervasive
636                             tProcedure      tThen           tType
637                             tVar            }
638         {   4682    }       0,      8,    32,  6409, 24632,  1040,  8210,
639                             { tNewLine      tSemicolon      tAssert
640                             tBind           tChecked        tConst
641                             tConverter      tElse           tElseif
642                             tEnd            tFunction       tIf
643                             tInline         tNot            tPervasive
644                             tProcedure      tType           tVar
645                             }
646         {   4690    }       0,      0,    32,  6409, 24632,  1040,  8210,
647                             { tSemicolon    tAssert         tBind
648                             tChecked        tConst          tConverter
649                             tElse           tElseif         tEnd
650                             tFunction       tIf             tInline
651                             tNot            tPervasive      tProcedure
652                             tType           tVar            }
653         {   4698    }       0,      0,     0,     0, 16440,     0,     0,
654                             { tElse         tElseif         tEnd
655                             tIf             }
656         {   4706    }       0,      0,     0,     0,    24,     0,     0,
657                             { tElse         tElseif         }
658         {   4714    }       0,      0,     0,     0, 16416,     0,     0,
659                             { tEnd          tIf             }
660         {   4722    }       0,      0,     0,     0, 16384,     0,     0,
661                             { tIf           }
662         {   4730    }     723, 32744,  4128,  6409,  8220,  5394,  9234,
663                             { tIdent        tNumber         tString
664                             tMDString       tChar           tMDChar
665                             tNewLine        tPlus           tMinus
666                             tAsterisk       tEqual          tLessThan
667                             tGreaterThan    tLessEqual      tGreaterEqual
668                             tImplies        tLeftParen      tSemicolon
669                             tAnd            tAssert         tBind
670                             tChecked        tConst          tConverter
671                             tDiv            tElse           tElseif
672                             tFunction       tIn             tInline
673                             tMod            tNot            tOr
674                             tPervasive      tProcedure      tThen
675                             tType           tVar            tXor
676                             }
677         {   4738    }       0,      8,    32,  6409,  8216,  1040,  9234,
678                             { tNewLine      tSemicolon      tAssert
679                             tBind           tChecked        tConst
680                             tConverter      tElse           tElseif
```

```
681                                 tFunction       tInline         tNot
682                                 tPervasive      tProcedure      tThen
683                                 tType           tVar             }
684         {   4746    }             0,     8,     32, 6409,   8216, 1040,   8210,
685                                 { tNewLine      tSemicolon      tAssert
686                                 tBind           tChecked        tConst
687                                 tConverter      tElse           tElseif
688                                 tFunction       tInline         tNot
689                                 tPervasive      tProcedure      tType
690                                 tVar             }
691         {   4754    }             0,     0,     32, 6409,   8216, 1040,   8210,
692                                 { tSemicolon    tAssert         tBind
693                                 tChecked        tConst          tConverter
694                                 tElse           tElseif         tFunction
695                                 tInline         tNot            tPervasive
696                                 tProcedure      tType           tVar
697                                  }
698         {   4762    }             0,     0,     32, 6409,   8224, 1104,   8210,
699                                 { tSemicolon    tAssert         tBind
700                                 tChecked        tConst          tConverter
701                                 tEnd            tFunction       tInline
702                                 tLoop           tNot            tPervasive
703                                 tProcedure      tType           tVar
704                                  }
705         {   4770    }             0,     0,      0,     0,    32,    64,      0,
706                                 { tEnd          tLoop            }
707         {   4778    }             0,     0,      0,     0,    0,     64,      0,
708                                 { tLoop          }
709         {   4786    }           723, 32744,  4100, 16384,     4,  5442,      0,
710                                 { tIdent        tNumber         tString
711                                 tMDString       tChar           tMDChar
712                                 tNewLine        tPlus           tMinus
713                                 tAsterisk       tEqual          tLessThan
714                                 tGreaterThan    tLessEqual      tGreaterEqual
715                                 tImplies        tLeftParen      tDoublePeriod
716                                 tAnd            tDecreasing     tDiv
717                                 tIn             tLoop           tMod
718                                 tNot            tOr             tXor
719                                  }
720         {   4794    }           723, 32744,  4100,     0,     4,  5442,      0,
721                                 { tIdent        tNumber         tString
722                                 tMDString       tChar           tMDChar
723                                 tNewLine        tPlus           tMinus
724                                 tAsterisk       tEqual          tLessThan
725                                 tGreaterThan    tLessEqual      tGreaterEqual
726                                 tImplies        tLeftParen      tDoublePeriod
727                                 tAnd            tDiv            tIn
728                                 tLoop           tMod            tNot
729                                 tOr             tXor             }
730         {   4802    }           723, 16616,     4,     0,     4,   320,      0,
731                                 { tIdent        tNumber         tString
732                                 tMDString       tChar           tMDChar
```

```
733                          tNewLine    tPlus       tMinus
734                          tAsterisk   tLeftParen  tDoublePeriod
735                          tDiv        tLoop       tMod
736                          tXor         }
737     {  4810  }            0,    8,    0,    0,    0,   64,    0,
738                          { tNewLine    tLoop       }
739     {  4818  }           723, 32744, 4510, 2080,   36, 15618, 4128,
740                          { tIdent    tNumber     tString
741                          tMDString   tChar       tMDChar
742                          tNewLine    tPlus       tMinus
743                          tAsterisk   tEqual      tLessThan
744                          tGreaterThan tLessEqual tGreaterEqual
745                          tImplies    tLeftParen  tPeriod
746                          tDoublePeriod tAssign   tComma
747                          tUpArrow    tLabel      tAnd
748                          tBound      tConst      tDiv
749                          tEnd        tIn         tMod
750                          tNot        tOf         tOr
751                          tOtherwise  tReadonly   tTo
752                          tVar        tXor         }
753     {  4826  }           723, 32744, 4502,   32,   36, 15618, 4096,
754                          { tIdent    tNumber     tString
755                          tMDString   tChar       tMDChar
756                          tNewLine    tPlus       tMinus
757                          tAsterisk   tEqual      tLessThan
758                          tGreaterThan tLessEqual tGreaterEqual
759                          tImplies    tLeftParen  tPeriod
760                          tDoublePeriod tComma    tUpArrow
761                          tLabel      tAnd        tBound
762                          tDiv        tEnd        tIn
763                          tMod        tNot        tOf
764                          tOr         tOtherwise  tTo
765                          tXor         }
766     {  4834  }           723, 32744, 4502,    0,   36, 15618, 4096,
767                          { tIdent    tNumber     tString
768                          tMDString   tChar       tMDChar
769                          tNewLine    tPlus       tMinus
770                          tAsterisk   tEqual      tLessThan
771                          tGreaterThan tLessEqual tGreaterEqual
772                          tImplies    tLeftParen  tPeriod
773                          tDoublePeriod tComma    tUpArrow
774                          tLabel      tAnd        tDiv
775                          tEnd        tIn         tMod
776                          tNot        tOf         tOr
777                          tOtherwise  tTo         tXor
778                           }
779     {  4842  }           723, 32744, 4502,    0,   36, 15618,    0,
780                          { tIdent    tNumber     tString
781                          tMDString   tChar       tMDChar
782                          tNewLine    tPlus       tMinus
783                          tAsterisk   tEqual      tLessThan
784                          tGreaterThan tLessEqual tGreaterEqual
```

```
785                     tImplies        tLeftParen      tPeriod
786                     tDoublePeriod   tComma          tUpArrow
787                     tLabel          tAnd            tDiv
788                     tEnd            tIn             tMod
789                     tNot            tOf             tOr
790                     tOtherwise      tXor             }
791     {  4850  }       723, 32744,    4372,    0,     36, 15618,     0,
792                     { tIdent          tNumber         tString
793                     tMDString       tChar           tMDChar
794                     tNewLine        tPlus           tMinus
795                     tAsterisk       tEqual          tLessThan
796                     tGreaterThan    tLessEqual      tGreaterEqual
797                     tImplies        tLeftParen      tDoublePeriod
798                     tComma          tLabel          tAnd
799                     tDiv            tEnd            tIn
800                     tMod            tNot            tOf
801                     tOr             tOtherwise      tXor
802                      }
803     {  4858  }       723, 32744,    4510,    0,     36, 15618,     0,
804                     { tIdent          tNumber         tString
805                     tMDString       tChar           tMDChar
806                     tNewLine        tPlus           tMinus
807                     tAsterisk       tEqual          tLessThan
808                     tGreaterThan    tLessEqual      tGreaterEqual
809                     tImplies        tLeftParen      tPeriod
810                     tDoublePeriod   tAssign         tComma
811                     tUpArrow        tLabel          tAnd
812                     tDiv            tEnd            tIn
813                     tMod            tNot            tOf
814                     tOr             tOtherwise      tXor
815                      }
816     {  4866  }       723, 32744,    4510,   32,     36, 15618,  4096,
817                     { tIdent          tNumber         tString
818                     tMDString       tChar           tMDChar
819                     tNewLine        tPlus           tMinus
820                     tAsterisk       tEqual          tLessThan
821                     tGreaterThan    tLessEqual      tGreaterEqual
822                     tImplies        tLeftParen      tPeriod
823                     tDoublePeriod   tAssign         tComma
824                     tUpArrow        tLabel          tAnd
825                     tBound          tDiv            tEnd
826                     tIn             tMod            tNot
827                     tOf             tOr             tOtherwise
828                     tTo             tXor             }
829     {  4874  }       723, 32744,    4372,    0,     36, 13570,     0,
830                     { tIdent          tNumber         tString
831                     tMDString       tChar           tMDChar
832                     tNewLine        tPlus           tMinus
833                     tAsterisk       tEqual          tLessThan
834                     tGreaterThan    tLessEqual      tGreaterEqual
835                     tImplies        tLeftParen      tDoublePeriod
836                     tComma          tLabel          tAnd
```

```
837                                 tDiv            tEnd            tIn
838                                 tMod            tNot            tOr
839                                 tOtherwise      tXor             }
840         {   4882    }             723, 32736,   4372,     0,     36, 13570,     0,
841                                 { tIdent          tNumber         tString
842                                 tMDString       tChar           tMDChar
843                                 tPlus           tMinus          tAsterisk
844                                 tEqual          tLessThan       tGreaterThan
845                                 tLessEqual      tGreaterEqual   tImplies
846                                 tLeftParen      tDoublePeriod   tComma
847                                 tLabel          tAnd            tDiv
848                                 tEnd            tIn             tMod
849                                 tNot            tOr             tOtherwise
850                                 tXor             }
851         {   4890    }             723, 32744,   4404, 6473,   8228, 13586,   8210,
852                                 { tIdent          tNumber         tString
853                                 tMDString       tChar           tMDChar
854                                 tNewLine        tPlus           tMinus
855                                 tAsterisk       tEqual          tLessThan
856                                 tGreaterThan    tLessEqual      tGreaterEqual
857                                 tImplies        tLeftParen      tDoublePeriod
858                                 tComma          tSemicolon      tLabel
859                                 tAnd            tAssert         tBind
860                                 tCase           tChecked        tConst
861                                 tConverter      tDiv            tEnd
862                                 tFunction       tIn             tInline
863                                 tMod            tNot            tOr
864                                 tOtherwise      tPervasive      tProcedure
865                                 tType           tVar            tXor
866                                  }
867         {   4898    }               0,     8,     32, 6473,   8224, 1040,   8210,
868                                 { tNewLine        tSemicolon      tAssert
869                                 tBind           tCase           tChecked
870                                 tConst          tConverter      tEnd
871                                 tFunction       tInline         tNot
872                                 tPervasive      tProcedure      tType
873                                 tVar             }
874         {   4906    }               0,     0,     32, 6473,   8224, 1040,   8210,
875                                 { tSemicolon      tAssert         tBind
876                                 tCase           tChecked        tConst
877                                 tConverter      tEnd            tFunction
878                                 tInline         tNot            tPervasive
879                                 tProcedure      tType           tVar
880                                  }
881         {   4914    }               0,     0,     0,    64,    32,     0,     0,
882                                 { tCase           tEnd             }
883         {   4922    }               0,     0,     0,    64,    0,      0,     0,
884                                 { tCase          }
885         {   4930    }             723, 24544,   4096,     0,     4, 5378,     0,
886                                 { tIdent          tNumber         tString
887                                 tMDString       tChar           tMDChar
888                                 tPlus           tMinus          tAsterisk
```

```
889                                 tEqual          tLessThan       tGreaterThan
890                                 tLessEqual      tGreaterEqual   tLeftParen
891                                 tAnd            tDiv            tIn
892                                 tMod            tNot            tOr
893                                 tXor             }
894         {   4938    }            723, 24544,  4096,     0,       4,  1282,     0,
895                                 { tIdent        tNumber         tString
896                                 tMDString       tChar           tMDChar
897                                 tPlus           tMinus          tAsterisk
898                                 tEqual          tLessThan       tGreaterThan
899                                 tLessEqual      tGreaterEqual   tLeftParen
900                                 tAnd            tDiv            tIn
901                                 tMod            tNot            tXor
902                                  }
903         {   4946    }            723, 24544,     0,     0,       4,  1282,     0,
904                                 { tIdent        tNumber         tString
905                                 tMDString       tChar           tMDChar
906                                 tPlus           tMinus          tAsterisk
907                                 tEqual          tLessThan       tGreaterThan
908                                 tLessEqual      tGreaterEqual   tLeftParen
909                                 tDiv            tIn             tMod
910                                 tNot            tXor             }
911         {   4954    }            723, 24544,     4,     0,       4,  1282,     0,
912                                 { tIdent        tNumber         tString
913                                 tMDString       tChar           tMDChar
914                                 tPlus           tMinus          tAsterisk
915                                 tEqual          tLessThan       tGreaterThan
916                                 tLessEqual      tGreaterEqual   tLeftParen
917                                 tDoublePeriod   tDiv            tIn
918                                 tMod            tNot            tXor
919                                  }
920         {   4962    }            723, 16864,     4,     0,       4,   258,     0,
921                                 { tIdent        tNumber         tString
922                                 tMDString       tChar           tMDChar
923                                 tPlus           tMinus          tAsterisk
924                                 tEqual          tLeftParen      tDoublePeriod
925                                 tDiv            tIn             tMod
926                                 tXor             }
927         {   4970    }              0,   128,     0,     0,       4,   256,     0,
928                                 { tAsterisk     tDiv            tMod
929                                  }
930         {   4978    }           1023, 32736,  4227,     0,       4,  5378,     0,
931                                 { tIdent        tNumber         tOctalNumber
932                                 tHexNumber      tString         tExString
933                                 tMDString       tChar           tExChar
934                                 tMDChar         tPlus           tMinus
935                                 tAsterisk       tEqual          tLessThan
936                                 tGreaterThan    tLessEqual      tGreaterEqual
937                                 tImplies        tLeftParen      tRightParen
938                                 tPeriod         tUpArrow        tAnd
939                                 tDiv            tIn             tMod
940                                 tNot            tOr             tXor
```

```
941                          }
942         {   4986   }       723, 32736, 14487,     0,     4,  5378, 16385,
943                          { tIdent         tNumber         tString
944                          tMDString       tChar           tMDChar
945                          tPlus           tMinus          tAsterisk
946                          tEqual          tLessThan       tGreaterThan
947                          tLessEqual      tGreaterEqual   tImplies
948                          tLeftParen      tRightParen     tPeriod
949                          tDoublePeriod   tComma          tUpArrow
950                          tAll            tAnd            tAny
951                          tDiv            tIn             tMod
952                          tNot            tOr             tParameter
953                          tUnknown        tXor             }
954         {   4994   }       723, 32736, 14357,     0,     4,  5378, 16385,
955                          { tIdent         tNumber         tString
956                          tMDString       tChar           tMDChar
957                          tPlus           tMinus          tAsterisk
958                          tEqual          tLessThan       tGreaterThan
959                          tLessEqual      tGreaterEqual   tImplies
960                          tLeftParen      tRightParen     tDoublePeriod
961                          tComma          tAll            tAnd
962                          tAny            tDiv            tIn
963                          tMod            tNot            tOr
964                          tParameter      tUnknown        tXor
965                           }
966         {   5002   }         0,     0,    17,     0,     0,     0,     0,
967                          { tRightParen    tComma           }
968         {   5010   }         0);
969
970
971             { Table Walker State }
972
973             var processing:
974                 Boolean := true;
975             var tracing:
976                 Boolean := false;
977             var semanticizing:
978                 Boolean := true;
979             var tablePointer:
980                 0 .. tableSize := 0;
981             var operation:
982                 firstTableOperation .. lastTableOperation;
983             const dummyLine := 0;
984
985
986             { The Call Stack:
987
988              The Call Stack implements Syntax/Semantic
989              Language procedure call and return.  In addition,
990              the Call Stack contains dynamic error recovery sets.
991              Each time an oCall operation is executed,
992              the table return address and the reachable set
```

```
993             for the oCall operator are pushed onto the
994             Call Stack.  When an oReturn is executed, the
995             return address and the reachable set are popped from the
996             Stack. An oReturn executed when the Call stack is
997             empty terminates table execution.          }
998
999           const setSize := 7;
000           type SetType =
001               array 0 .. setSize of PowerSet;
002
003           type CallStackEntry =
004               record
005                   var sett:
006                       SetType;  { Dynamic set of syntax recovery tokens}
007                   var returnAddress:
008                       0 .. tableSize;
009               end CallStackEntry;
010
011           const callStackSize := 127;
012           var callStack:
013               array 0 .. callStackSize of CallStackEntry;
014           var callStackTop:
015               0 .. callStackSize := 0;
016
017
018
019           { Choice Match Flag:
020
021             Set by the Choice Handler to indicate whether
022             a match was made or the otherwise path was taken.
023             Set to true if a match was made and false otherwise.
024             This flag is used in input choices to indicate
025             whether the choice input token should be accepted or
026             not.                                          }
027
028           var choiceTagMatched:
029               Boolean;
030
031
032           { Parameterized And Choice Semantic Operation Values:
033
034             These are used to hold the decoded parameter value to
035             a parameterized semantic operation and the result
036             value returned by a choice semantic operation
037             respectively.                                 }
038
039           var parameterValue:
040               SignedInt;
041           var resultValue:
042               SignedInt;
043
044
```

```
045             { Line Counter }
046
047             pervasive const maxLineNumber := 29999;
048
049             var lineNumber:
050                 0 .. maxLineNumber := 1;
051
052
053             { Input/output channel number assignments. }
054
055             const outputFile := 1;  {Output token stream.}
056             const errorFile := 2;   {Output error stream.}
057             const inputFile := 3;   {Input token stream (produced by
058                                                          Screener) }
059
060
061             { Predefined flag values for Powerset module }
062
063             const maxBits := 15;
064             const flagValue:
065                 array 0 .. maxBits of SignedInt :=
066                     (1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048,
067                      4096, 8192, 16384, 1 { unused });
068
069
070             { Error Output Interface }
071
072             pervasive const firstErrorCode := 0;
073             pervasive const eNoError := 0;
074             pervasive const eEndOfErrors := 1;
075             pervasive const firstTrueErrorCode := 2;
076             pervasive const firstParserErrorCode := 21;
077             pervasive const eSyntaxError := 21;
078             pervasive const firstFatalErrorCode := 22;
079             pervasive const ePrematureEndOfFile := 22;
080             pervasive const eCallStackOverflow := 23;
081             pervasive const eExtraneousProgramText := 24;
082             pervasive const lastErrorCode := 24;
083
084             const maxErrors := 20;
085             var noErrors:
086                 0 .. maxErrors := 0;
087
088
089
090             procedure EmitError (errorCode: firstErrorCode ..
091                                                 lastErrorCode,
092                                  lineNo: SignedInt) =
093
094                 imports (var IO, errorFile, lineNumber, var noErrors,
095                          var processing, tracing);
096
```

```
097             { This procedure provides output of error
098              codes to the error message lister;  all
099              error codes are output using this procedure.     }
100
101             begin
102                 if errorCode >= firstTrueErrorCode then
103                     noErrors := noErrors + 1;
104                 end if;
105
106                 { Write lineNumber, errorCode, lineNo (S/SL) to
107                   ErrorFile }
108
109                 IO.WriteInt (errorFile, lineNumber);
110                 IO.WriteInt (errorFile, errorCode);
111                 IO.WriteInt (errorFile, lineNo);
112
113                 if errorCode >= firstFatalErrorCode
114                     noErrors = maxErrors then
115                     processing := false;
116                 end if;
117
118                 { Trace error emission }
119
120                 if tracing then
121                     IO.PutString ('Error output ');
122                     IO.PutInt (errorCode);
123                     IO.PutString ('; Line ');
124                     IO.PutLine;
125                 end if;
126
127             end EmitError;
128
129
130         { Input Interface }
131
132         pervasive const firstInputToken := -1;
133         pervasive const tSyntaxError := -1;
134         pervasive const tIdent := 0;
135         pervasive const tNumber := 1;
136         pervasive const tOctalNumber := 2;
137         pervasive const tHexNumber := 3;
138         pervasive const tString := 4;
139         pervasive const tExString := 5;
140         pervasive const tMDString := 6;
141         pervasive const tChar := 7;
142         pervasive const tExChar := 8;
143         pervasive const tMDChar := 9;
144         pervasive const tComment := 10;
145         pervasive const tCommentStart := 11;
146         pervasive const tCommentMiddle := 12;
147         pervasive const tCommentEnd := 13;
148         pervasive const tCodeStart := 14;
```

```
149             pervasive const tCodeMiddle := 15;
150             pervasive const tCodeEnd := 16;
151             pervasive const tIllegal := 17;
152             pervasive const tNewLine := 18;
153             pervasive const tEndOfFile := 19;
154             pervasive const tPlus := 20;
155             pervasive const tMinus := 21;
156             pervasive const tAsterisk := 22;
157             pervasive const tEqual := 23;
158             pervasive const tLessThan := 24;
159             pervasive const tGreaterThan := 25;
160             pervasive const tLessEqual := 26;
161             pervasive const tGreaterEqual := 27;
162             pervasive const tImplies := 28;
163             pervasive const tLeftParen := 29;
164             pervasive const tRightParen := 30;
165             pervasive const tPeriod := 31;
166             pervasive const tDoublePeriod := 32;
167             pervasive const tAssign := 33;
168             pervasive const tComma := 34;
169             pervasive const tSemicolon := 35;
170             pervasive const tColon := 36;
171             pervasive const tUpArrow := 37;
172             pervasive const tLabel := 38;
173             pervasive const tAbstraction := 39;
174             pervasive const tAligned := 40;
175             pervasive const tAll := 41;
176             pervasive const tAnd := 42;
177             pervasive const tAny := 43;
178             pervasive const tArray := 44;
179             pervasive const tAssert := 45;
180             pervasive const tAt := 46;
181             pervasive const tBegin := 47;
182             pervasive const tBind := 48;
183             pervasive const tBits := 49;
184             pervasive const tBound := 50;
185             pervasive const tCase := 51;
186             pervasive const tCheckable := 52;
187             pervasive const tChecked := 53;
188             pervasive const tCode := 54;
189             pervasive const tCollection := 55;
190             pervasive const tConst := 56;
191             pervasive const tConverter := 57;
192             pervasive const tCounted := 58;
193             pervasive const tDecreasing := 59;
194             pervasive const tDefault := 60;
195             pervasive const tDependent := 61;
196             pervasive const tDiv := 62;
197             pervasive const tElse := 63;
198             pervasive const tElseif := 64;
199             pervasive const tEnd := 65;
200             pervasive const tExit := 66;
```

```
201             pervasive const tExports := 67;
202             pervasive const tExternal := 68;
203             pervasive const tFinally := 69;
204             pervasive const tFor := 70;
205             pervasive const tForward := 71;
206             pervasive const tFrom := 72;
207             pervasive const tFunction := 73;
208             pervasive const tIf := 74;
209             pervasive const tImports := 75;
210             pervasive const tIn := 76;
211             pervasive const tInclude := 77;
212             pervasive const tInitially := 78;
213             pervasive const tInline := 79;
214             pervasive const tInvariant := 80;
215             pervasive const tLoop := 81;
216             pervasive const tMachine := 82;
217             pervasive const tMod := 83;
218             pervasive const tModule := 84;
219             pervasive const tNot := 85;
220             pervasive const tOf := 86;
221             pervasive const tOr := 87;
222             pervasive const tOtherwise := 88;
223             pervasive const tPacked := 89;
224             pervasive const tParameter := 90;
225             pervasive const tPervasive := 91;
226             pervasive const tPost := 92;
227             pervasive const tPre := 93;
228             pervasive const tProcedure := 94;
229             pervasive const tReadonly := 95;
230             pervasive const tRecord := 96;
231             pervasive const tReturn := 97;
232             pervasive const tReturns := 98;
233             pervasive const tSet := 99;
234             pervasive const tThen := 100;
235             pervasive const tThus := 101;
236             pervasive const tTo := 102;
237             pervasive const tType := 103;
238             pervasive const tUnknown := 104;
239             pervasive const tVar := 105;
240             pervasive const tWhen := 106;
241             pervasive const tWith := 107;
242             pervasive const tXor := 108;
243             pervasive const lastInputToken := tXor;
244
245             pervasive const firstTextInputToken := tIdent;
246             pervasive const lastTextInputToken := tIllegal;
247
248             pervasive const firstValueInputToken := tIdent;
249             pervasive const lastValueInputToken := tMDChar;
250
251
252             var nextInputToken:
```

```
253             firstInputToken  ..  lastInputToken;
254
255         const maxTokenLength := 500;
256         const maxStringLiteralLength := 500;
257
258         var inputToken:
259             firstInputToken .. lastInputToken;
260         var inputTokenLength:
261             1 .. maxTokenLength;
262         var inputTokenText: packed array 1..maxTokenLength of Char;
263
264         const maxIdents := 2000;
265
266         { Constant declarations for value representation field of
267           number tokens. }
268
269         pervasive const unsignedSingleInt := -1;
270         pervasive const signedSingleInt := 1;
271
272         type InputTokenValueType =
273             record
274                 var valueToken:
275                     firstValueInputToken .. lastValueInputToken;
276                 var intValueRepresentation:
277                     SignedInt;
278                 var integerValue:
279                     SignedInt;
280                 var identIndex:
281                     0 .. maxIdents - 1;
282                 var unnormalizedIdentIndex:
283                     0 .. maxIdents - 1;
284                 var stringLength:
285                     0 .. maxStringLiteralLength;
286                 var stringText:
287                     packed array 1..maxStringLiteralLength of Char;
288             end InputTokenValueType;
289
290         var inputTokenValue:
291             InputTokenValueType;
292
293
294         { Variable Used in Syntax Error Recovery }
295
296         var newInputLine:
297             Boolean := false;
298
299
300         procedure AcceptInputToken =
301
302             imports (var nextInputToken, var inputToken,
303                 var inputTokenValue, var lineNumber, var IO,
304                 maxTokenLength, tracing, var newInputLine,
```

```
305                 var inputTokenLength, inputFile, var inputTokenText);
306
307             { This procedure provides the interface to the
308              previous pass;  it is reponsible for handling
309              all input including line number indicators and
310              the values and text associated with input tokens.  }
311
312             pre (nextInputToken not = tEndOfFile);
313
314             post (not IO.EndFile(inputFile));
315
316             begin
317                 var inputInt:
318                     SignedInt;
319                 var dummyLength:
320                     1 .. maxTokenLength;
321                 var dummyText:
322                     packed array 1..maxTokenLength of Char;
323
324                 { Accept Token }
325
326                 inputToken := nextInputToken;
327
328
329                 { Read Value Associated with Token }
330
331                 if inputToken >= firstValueInputToken and
332                         inputToken <= lastValueInputToken then
333                     inputTokenValue.valueToken := inputToken;
334
335                     case inputToken of
336
337                         tIdent =>
338                             IO.ReadInt (inputFile, inputInt);
339                             inputTokenValue.identIndex := inputInt;
340                             IO.ReadInt (inputFile, inputInt);
341                             inputTokenValue.unnormalizedIdentIndex :=
342                                                         inputInt;
343                             end tIdent;
344
345                         tNumber, tOctalNumber, tHexNumber =>
346                             IO.ReadInt (inputFile, inputInt);
347                             assert (inputInt = signedSingleInt or
348                                   inputInt = unsignedSingleInt);
349                             inputTokenValue.intValueRepresentation :=
350                                                       inputInt;
351                             IO.ReadInt (inputFile, inputInt);
352                             inputTokenValue.integerValue := inputInt;
353                             end tNumber {, tOctalNumber, tHexNumber };
354
355                         tString, tExString, tMDString, tChar, tExChar,
356                                                       tMDChar =>
```

```
357                     IO.ReadInt (inputFile, inputInt);
358                     inputTokenValue.stringLength := inputInt;
359
360                     if inputTokenValue.stringLength not = 0
361                         then
362                         IO.ReadString(inputFile,
363                             inputTokenValue.stringText,
364                             inputTokenValue.stringLength,
365                                                 inputInt);
366                         assert (inputInt =
367                                 inputTokenValue.stringLength);
368                     end if;
369
370                     end tString {, tExString, tMDString,
371                                    tChar, tExChar, tMDChar }
372             end case;
373         end if;
374
375
376         { Read Text of Token }
377
378         if inputToken >= firstTextInputToken and
379                 inputToken <= lastTextInputToken then
380             IO.ReadInt (inputFile, inputInt);
381             inputTokenLength := inputInt;
382             IO.ReadString (inputFile, inputTokenText,
383                   inputTokenLength, inputInt);
384             assert (inputInt = inputTokenLength);
385         end if;
386
387
388         { Read Next Input Token Number }
389
390         newInputLine := false;
391
392         loop
393             IO.ReadInt (inputFile, inputInt);
394             nextInputToken := inputInt;
395
396             { Skip New Lines and Comments }
397
398             exit when nextInputToken not = tNewLine and
399                 nextInputToken not = tCommentStart and
400                 nextInputToken not = tCommentMiddle and
401                 nextInputToken not = tCommentEnd and
402                 nextInputToken not = tComment;
403
404             if nextInputToken = tNewLine then
405                 { Update Line Counter and Set Flag }
406                 newInputLine := true;
407
408                 if lineNumber < maxLineNumber then
```

```
409                         lineNumber := lineNumber + 1;
410                     else
411                         lineNumber := 0;
412                     end if;
413                 else
414                     { Throw Away Text of Comment }
415                     IO.ReadInt (inputFile, inputInt);
416                     dummyLength := inputInt;
417                     IO.ReadString (inputFile, dummyText,
418                                    dummyLength, inputInt);
419                     assert (inputInt = dummyLength);
420                 end if;
421
422             end loop;
423
424
425             { Trace input }
426
427             if tracing then
428                 IO.PutString ('Input token accepted ');
429                 IO.PutInt (inputToken);
430                 IO.PutString ('; Next input token ');
431                 IO.PutInt (nextInputToken);
432                 IO.PutString ('; Line ');
433                 IO.PutInt (lineNumber);
434                 IO.PutLine;
435             end if;
436
437         end AcceptInputToken;
438
439
440     { Output Interface }
441
442     pervasive const firstOutputToken := 0;
443     pervasive const aIdent := 0;
444     pervasive const aNumber := 1;
445     pervasive const aString := 2;
446     pervasive const aMDString := 3;
447     pervasive const aChar := 4;
448     pervasive const aMDChar := 5;
449     pervasive const aCodeUnit := 6;
450     pervasive const aStandardComponent := 7;
451     pervasive const aLegalitySpecifier := 8;
452     pervasive const aNewLine := 9;
453     pervasive const aEndOfFile := 10;
454     pervasive const aAbstraction := 11;
455     pervasive const aAdd := 12;
456     pervasive const aAligned := 13;
457     pervasive const aAll := 14;
458     pervasive const aAnd := 15;
459     pervasive const aAny := 16;
460     pervasive const aArray := 17;
```

```
461             pervasive const aAssert := 18;
462             pervasive const aAssign := 19;
463             pervasive const aAt := 20;
464             pervasive const aBegin := 21;
465             pervasive const aBind := 22;
466             pervasive const aBits := 23;
467             pervasive const aBound := 24;
468             pervasive const aCase := 25;
469             pervasive const aCheckable := 26;
470             pervasive const aChecked := 27;
471             pervasive const aCode := 28;
472             pervasive const aCollection := 29;
473             pervasive const aConst := 30;
474             pervasive const aConstType := 31;
475             pervasive const aConverter := 32;
476             pervasive const aCountMax := 33;
477             pervasive const aCounted := 34;
478             pervasive const aDecreasing := 35;
479             pervasive const aDefault := 36;
480             pervasive const aDiv := 37;
481             pervasive const aElse := 38;
482             pervasive const aEndBegin := 39;
483             pervasive const aEndBind := 40;
484             pervasive const aEndCase := 41;
485             pervasive const aEndCode := 42;
486             pervasive const aEndEnum := 43;
487             pervasive const aEndExports := 44;
488             pervasive const aEndExpression := 45;
489             pervasive const aEndIdent := 46;
490             pervasive const aEndIf := 47;
491             pervasive const aEndImports := 48;
492             pervasive const aEndLabel := 49;
493             pervasive const aEndLabels := 50;
494             pervasive const aEndLoop := 51;
495             pervasive const aEndModule := 52;
496             pervasive const aEndParms := 53;
497             pervasive const aEndRecord := 54;
498             pervasive const aEndSubs := 55;
499             pervasive const aEndValues := 56;
500             pervasive const aEndWith := 57;
501             pervasive const aEnum := 58;
502             pervasive const aEqual := 59;
503             pervasive const aExit := 60;
504             pervasive const aExports := 61;
505             pervasive const aExternal := 62;
506             pervasive const aField := 63;
507             pervasive const aFinally := 64;
508             pervasive const aFor := 65;
509             pervasive const aForward := 66;
510             pervasive const aFunction := 67;
511             pervasive const aGreater := 68;
512             pervasive const aGreaterEqual := 69;
```

```
513             pervasive const aIf := 70;
514             pervasive const aImply := 71;
515             pervasive const aImports := 72;
516             pervasive const aIn := 73;
517             pervasive const aInfixAnd := 74;
518             pervasive const aInfixCompare := 75;
519             pervasive const aInfixImply := 76;
520             pervasive const aInfixOr := 77;
521             pervasive const aInitial := 78;
522             pervasive const aInitially := 79;
523             pervasive const aInline := 80;
524             pervasive const aInvariant := 81;
525             pervasive const aLabels := 82;
526             pervasive const aLess := 83;
527             pervasive const aLessEqual := 84;
528             pervasive const aLoop := 85;
529             pervasive const aMDModule := 86;
530             pervasive const aMDRecord := 87;
531             pervasive const aMinus := 88;
532             pervasive const aMod := 89;
533             pervasive const aModule := 90;
534             pervasive const aModuleIdent := 91;
535             pervasive const aMultiply := 92;
536             pervasive const aNot := 93;
537             pervasive const aNotChecked := 94;
538             pervasive const aNotEqual := 95;
539             pervasive const aNotIn := 96;
540             pervasive const aOr := 97;
541             pervasive const aOtherwise := 98;
542             pervasive const aPacked := 99;
543             pervasive const aParameter := 100;
544             pervasive const aParens := 101;
545             pervasive const aParmType := 102;
546             pervasive const aParms := 103;
547             pervasive const aPervasive := 104;
548             pervasive const aPointer := 105;
549             pervasive const aPost := 106;
550             pervasive const aPre := 107;
551             pervasive const aProcedure := 108;
552             pervasive const aReadonly := 109;
553             pervasive const aRecord := 110;
554             pervasive const aReturn := 111;
555             pervasive const aReturnValue := 112;
556             pervasive const aReturns := 113;
557             pervasive const aSet := 114;
558             pervasive const aSubs := 115;
559             pervasive const aSubtract := 116;
560             pervasive const aTo := 117;
561             pervasive const aType := 118;
562             pervasive const aTypeName := 119;
563             pervasive const aUnknown := 120;
564             pervasive const aValues := 121;
```

```
565         pervasive const aVar := 122;
566         pervasive const aVarType := 123;
567         pervasive const aWhen := 124;
568         pervasive const aWith := 125;
569         pervasive const aXor := 126;
570         pervasive const lastOutputToken := aXor;
571
572
573         procedure EmitOutputToken (outputToken: firstOutputToken ..
574                                                 lastOutputToken) =
575
576             imports (var IO, outputFile, tracing);
577
578             pre { outputFile is open };
579
580             begin
581                 IO.WriteInt (outputFile, outputToken);
582
583                 { Trace output }
584
585                 if tracing then
586                     IO.PutString ('Output token ');
587                     IO.PutInt (outputToken);
588                     IO.PutLine;
589                 end if;
590
591             end EmitOutputToken;
592
593
594         function TokenInSet (keys: SetType)
595                 returns membership: Boolean =
596
597             imports (nextInputToken, setSize, SetIn);
598
599             { This function determines if the nextInputToken is in the
600               recovery set of tokens }
601
602             begin
603                 var wordNo:
604                     0 .. setSize := 0;
605                 var found:
606                     Boolean := false;
607
608                 loop
609                     if SetIn (keys (wordNo), flagValue
610                                 (nextInputToken)) then
611                         found := true;
612                     end if;
613                     exit when (found) or (wordNo = setSize);
614
615                     wordNo := wordNo + 1;
616                 end loop;
```

```
617
618                 membership := found;
619
620             end TokenInSet;
621
622
623
624         procedure Check (lineNo: SignedInt, keys: SetType) =
625
626             imports (TokenInSet, EmitError,eSyntaxError);
627
628             { This procedure checks if the incoming token is in the
629               current recovery token set }
630
631             begin
632
633                 if not TokenInSet (keys) then
634                     EmitError (eSyntaxError, lineNo);
635                 end if;
636
637             end Check;
638
639
640         procedure Union (var keys: SetType, keySet: SetType) =
641
642             imports (maxBits, SetAdd, PowerSet, setSize, callStack,
643                      callStackTop, var SetAdd, SetIn);
644
645             { This procedure produces the union of 2 sets }
646
647             begin
648                 var count:
649                     0 .. setSize + 1 := 0;
650                 var bitNo:
651                     0 .. maxBits;
652                 var wordNo:
653                     PowerSet;  { 15-bit portion of set }
654
655                 keys := callStack (callStackTop).sett;
656
657                 loop
658                     exit when (count > setSize);
659
660                     wordNo := keySet (count);
661                     bitNo := 0;
662
663                     loop
664                         exit when (bitNo = maxBits);
665
666                         if SetIn (wordNo, flagValue (bitNo)) then
667                             SetAdd (keys (count), flagValue (bitNo));
668                         end if;
```

```
669                     bitNo := bitNo + 1;
670                 end loop;
671
672                 count := count + 1;
673             end loop;
674
675         end Union;
676
677
678
679     procedure ConvertToKeySet (var address: SignedInt,
680                                var keySet: SetType) =
681
682         imports (Table, setSize);
683
684         { This procedure converts a tableAddress to a static key
685           set }
686
687         begin
688             var wordNo:
689                 0 .. setSize := 0;
690
691             loop
692                 keySet (wordNo) := table (address);
693                 exit when (wordNo = setSize);
694
695                 address := address + 1;
696             end loop;
697
698         end ConvertToKeySet;
699
700
701     procedure SkipTokens (keys: SetType) =
702
703         imports (TokenInSet, processing, EmitError, nextInputToken,
704                  tEndOfFile, AcceptInputToken);
705
706         { This procedure skips over inputTokens until one matches
707           a token in the recovery token set }
708
709         begin
710
711             loop
712                 exit when (TokenInSet (keys));
713                 AcceptInputToken;
714             end loop;
715
716         end SkipTokens;
717
718
719     { The Choice Handler }
720
```

```
721         procedure HandleChoice (choiceTag: SignedInt) =
722
723             imports (table, operation, var tablePointer,
724                      ConvertToKeySet, Union, Check, SkipTokens,
725
726                       var choiceTagMatched, SetType);
727             pre (operation = oInputChoice or
728                 (operation >= firstChoiceOperation and
729                     operation <= lastChoiceOperation));
730
731             { This procedure performs both input and semantic
732              choices.  It sequentially tests each alternative
733              value against the tag value, and when a match is
734              found, performs a branch to the corresponding
735              alternative path.  If none of the alternative
736              values matches the tag value, table interpretation
737              proceeds to the operation immediately following
738              the list of alternatives (normally the otherwise
739              path).  The flag choiceTagMatched is set to true
740              if a match is found and false otherwise.    }
741
742             begin
743                 var numberOfChoices:
744                     SignedInt;
745                 var tableAddress:
746                     SignedInt;
747                 var keySet:
748                     SetType;
749                 var lineNo:
750                     SignedInt;
751                 var keyAddress:
752                     SignedInt;
753                 var keys:
754                     SetType;
755
756                 tableAddress := table(tablePointer);
757                 lineNo := table (tablePointer + 1);
758                 keyAddress := table (tablePointer + 2);
759                 numberOfChoices := table(tableAddress);
760                 tablePointer := tableAddress + 1;
761                 choiceTagMatched := false;
762                 ConvertToKeySet (keyAddress, keySet);
763                 Union (keys, keySet);
764                 Check (lineNo, keys);
765                 SkipTokens (keys);
766
767                 loop
768                     if table(tablePointer) = choiceTag then
769                         choiceTagMatched := true;
770                         tablePointer := table(tablePointer + 1);
771                         exit;
772                     end if;
```

```
773
774                     tablePointer := tablePointer + 2;
775                     numberOfChoices := numberOfChoices - 1;
776
777                     exit when numberOfChoices = 0;
778                 end loop;
779
780             end HandleChoice;
781
782
783         { Execution Tracing }
784
785         procedure Trace =
786
787             imports (operation, table, tablePointer, var IO);
788
789             begin
790                 IO.PutString ('Table index ');
791                 IO.PutInt (tablePointer-1);
792                 IO.PutString ('; Operation ');
793                 IO.PutInt (operation);
794                 IO.PutString ('; Argument ');
795                 IO.PutInt (table(tablePointer));
796                 IO.PutLine;
797
798             end Trace;
799
800
801
802         { Semantic Choice Failure }
803
804         procedure HandleSemanticChoiceFailure =
805
806             imports (lineNumber, Trace, var IO, Abort);
807
808             begin
809                 IO.PutString ('Semantic choice failed; Line ');
810                 IO.PutInt (lineNumber);
811                 IO.PutLine;
812                 Trace;
813                 Abort;
814
815             end HandleSemanticChoiceFailure;
816
817
818         { Syntax Error Handling }
819
820         procedure HandleSyntaxError =
821
822             imports (lineNumber, Union, table, tablePointer,
823                     callStack, callStackTop, EmitError, SkipTokens,
824                     ConvertToKeySet, eSyntaxError, SetType);
```

```
825
826             pre (operation = oInput);
827
828             { This procedure handles syntax errors in the input
829              to the Parser pass;
830
831              Syntax error recovery:
832              When a mismatch occurs between the the input token
833              and the syntax table, the following recovery is
834              employed.
835
836
837
838
839              The keyAddress is converted to keySet, and keys is
840              created by the union of keySet and the set in the top
841              callStack entry. An error message is emitted.  Incoming
842              tokens are skipped until one is found belonging in keys. }
843
844             begin
845                 var keyAddress:
846                     SignedInt;
847                 var lineNo:
848                     SignedInt;  { S/SL lineNumber }
849                 var keySet:
850                     SetType;
851                 var keys:
852                     SignedInt;
853
854                 lineNo := table (tablePointer + 1);
855                 keyAddress := table (tablePointer + 2);
856                 ConvertToKeySet (keyAddress, keySet);
857                 Union (keys, keySet);  { Merges top of callStack set
858                                          with static set }
859                 EmitError (eSyntaxError, lineNo);  { Emit error
860                                                      message }
861                 SkipTokens (keys);  { Read input until token in
862                                       recovery set }
863
864             end HandleSyntaxError;
865
866
867
868
869
870         var Semantic:
871             module
872
873                 imports (operation, parameterValue, var resultValue,
874                     inputToken, inputTokenLength, inputTokenText,
875                     lineNumber, EmitOutputToken, EmitError, var IO,
876                     inputTokenValue, outputFile);
```

```
877
878                 exports (SimpleOp, ParmOp, ChoiceOp, EmitOp);
879
880
881                 {   This module implements the semantic mechanisms
882                  used in the Syntax/Semantic Language for the pass.
883                  Normally the Parser will have no (or at least,
884                  very few) semantic operations.  The transliterator's
885                  Parser, however, will have to make type choices and
886                  emit ident information as text and hence
887                  will require semantic operations and data structures
888                  to support them. }
889
890
891                 { Declarations of constants, variables, data
892                  structures, procedures and modules used in
893                  implementing semantic mechanisms will go here. }
894
895
896                 procedure SimpleOp =
897
898                     imports (operation);
899
900                     pre (operation >= firstSimpleOperation and
901                         operation <= lastSimpleOperation);
902
903                     begin
904                         { No Simple Operations in the Parser }
905                         assert (false);
906
907                     end SimpleOp;
908
909
910
911                 procedure ParmOp =
912
913                     imports (operation, parameterValue);
914
915                     pre (operation >= firstParameterizedOperation and
916                         operation <= lastParameterizedOperation);
917
918                     begin
919                         { No Parameterized Operations in the Parser }
920                         assert (false);
921
922                     end ParmOp;
923
924
925
926                 procedure ChoiceOp =
927
928                     imports (operation, var resultValue);
```

```
929
930                 pre (operation >= firstChoiceOperation and
931                     operation <= lastChoiceOperation);
932
933                 begin
934                     { No Choice Operations in the Parser }
935                     assert (false);
936
937                 end ChoiceOp;
938
939
940             procedure EmitOp =
941
942                 imports (operation, inputToken, inputTokenLength,
943                     inputTokenValue, lineNumber, EmitOutputToken,
944                     var IO, outputFile, inputTokenText);
945
946                 pre (operation >= firstEmittingOperation and
947                     operation <= lastEmittingOperation
948                     { and outputFile is open } );
949
950                 begin
951
952                     case operation of
953
954                         oEmitIdent =>
955                             assert (inputTokenValue.valueToken =
956                                                         tIdent);
957                             IO.WriteInt (outputFile,
958                                         inputTokenValue.
959                                                     identIndex);
960                             IO.WriteInt (outputFile,
961                                       inputTokenValue.
962                                       unnormalizedIdentIndex);
963                             end oEmitIdent;
964
965                         oEmitChar, oEmitString =>
966                             assert (inputTokenValue.valueToken =
967                                                       tChar or
968                                 inputTokenValue.valueToken =
969                                                        tExChar or
970                                 inputTokenValue.valueToken =
971                                                         tMDChar or
972                                 inputTokenValue.valueToken =
973                                                       tString or
974                                 inputTokenValue.valueToken =
975                                                     tExSTring or
976                                 inputTokenValue.valueToken =
977                                                       tMDString);
978                             IO.WriteInt (outputFile,
979                                 inputTokenValue.stringLength);
980                             IO.WriteString (outputFile,
```

```
981                                 inputTokenValue.stringText,
982                                 inputTokenValue.stringLength);
983                             end oEmitChar {, oEmitString };
984
985                         oEmitNumber =>
986                             assert (inputTokenValue.valueToken =
987                                 tNumber or
988                                 inputTokenValue.valueToken =
989                                         tOctalNumber or
990                                 inputTokenValue.valueToken =
991                                         tHexNumber);
992                             IO.WriteInt (outputFile,
993                                 inputTokenValue.
994                                 intValueRepresentation);
995                             IO.WriteInt (outputFile,
996                                         inputTokenValue.
997                                         integerValue);
998                             end oEmitNumber;
999
000                         oEmitCode =>
001                             assert (inputToken = tCodeMiddle);
002                             IO.WriteInt (outputFile,
003                                             inputTokenLength);
004                             IO.WriteString (outputFile,
005                                 inputtokenText,
006                                 inputTokenLength);
007                             end oEmitCode;
008
009                         oEmitLine =>
010                             EmitOutputToken (aNewLine);
011                             IO.WriteInt (outputFile, lineNumber);
012                             end oEmitLine;
013
014                     end case;
015
016                 end EmitOp;
017
018
019             end module { Semantic };
020
021
022         { Main Walker Program }
023
024         initially
025
026             imports (table, var tablePointer, var processing,
027                 var operation, var callStack, var callStackTop, Trace,
028                 EmitError, var nextInputToken, var inputToken,
029                 HandleSyntaxError, HandleSemanticChoiceFailure,
030                 HandleChoice, EmitOutputToken, var resultValue,
031                 var parameterValue, var IO, inputFile, outputFile,
032                 errorFile, noErrors, Abort, ConvertToKeySet, Union,
```

```
033                 SetType, setSize, SetAdd, maxBits, var Semantic,
034                 semanticizing, var newInputLine, var tracing,
035                 var choiceTagMatched, AcceptInputtoken);
036
037             begin
038                 var inputInt:
039                     SignedInt;
040
041                 const maxOptionStringLength := 100;
042                 var optionString:
043                     packed array 1..maxOptionStringLength of Char;
044                 var optionStringLength:
045                     SignedInt;
046                 var keySet:
047                     SetType;
048                 var keys:
049                     SetType;
050                 var temp:
051                     SetType;
052                 var wordNo:
053                     0 .. setSize := 0;
054                 var bitNo:
055                     0 .. maxBits - 1;
056                 var address:
057                     SignedInt;
058                 var lineNo:
059                     SignedInt;
060
061
062                 { Process Options }
063
064                 IO.GetOptions (optionString, optionStringLength);
065
066                 if optionStringLength > 0 then
067                     if optionString(1) = $T or optionString(1) = $t
068                         then
069                         tracing := true;
070                     end if;
071                 end if;
072
073
074                 { Start Up I/O }
075
076                 IO.Open (inputFile, inFile);
077                 IO.Open (outputFile, outFile);
078                 IO.Open (errorFile, outFile);
079
080                 nextInputToken := tNewLine;
081                 AcceptInputToken;
082                 newInputLine := false;
083
084                 { Initialize set at bottom of callStack to hold only
```

```
085                   tEndOfFile token }
086
087                 temp := callStack (callStackTop).sett;
088
089                 loop
090                     SetClear (temp (wordNo));
091                     exit when (wordNo = setSize);
092                     wordNo := wordNo + 1;
093                 end loop;
094
095                 { Add tEndOfFile token }
096
097                 wordNo := tEndOfFile div maxBits;
098                 bitNo := tEndOfFile mod maxBits;
099                 SetAdd (temp (wordNo), flagValue (bitNo));
100                 callStack (callStackTop).sett := temp;
101
102                 { Walk the Syntax/Semantic Table }
103
104                 loop
105                     exit when not processing;
106
107                     operation := table(tablePointer);
108                     tablePointer := tablePointer + 1;
109
110                     { Trace Execution }
111                     if tracing then
112                         Trace;
113                     end if;
114
115                     case operation of
116
117                         oCall =>
118                             if callStackTop < callStackSize then
119                                 address := tablePointer + 1;
120                                 ConvertToKeySet (address, keySet);
121                                 { Merge set on top of callStack with
122                                    static keys }
123                                 Union (keys, keySet);
124                                 callStackTop := callStackTop + 1;
125                                 callStack (callStackTop).sett := keys;
126                                 callStack(callStackTop).returnAddress
127                                      := tablePointer + 2;
128                                 tablePointer := table(tablePointer);
129                             else
130                                 EmitError (eCallStackOverflow,
131                                                 dummyLine);
132                                 processing := false;
133                             end if;
134                             end oCall;
135
136                         oReturn =>
```

```
137                         if callStackTop = 0 then
138                             { Return from main S/SL procedure }
139                             processing := false;
140                         else
141                             tablePointer := callStack
142                                   (callStackTop).returnAddress;
143                             callStackTop := callStackTop - 1;
144                         end if;
145                         end oReturn;
146
147                     oRepeat, oMerge =>
148                         tablePointer := table(tablePointer);
149                         end oRepeat {, oMerge };
150
151                     oInput =>
152                         if table(tablePointer) = nextInputToken
153                             then
154                             AcceptInputToken;
155                         else
156                             { Syntax Error in Input }
157                             HandleSyntaxError;
158                         end if;
159
160                         tablePointer := tablePointer + 3;
161                         end oInput;
162
163                     oInputChoice =>
164                         HandleChoice (nextInputToken);
165
166                         if choiceTagMatched then
167                             AcceptInputToken;
168                         end if;
169                         end oInputChoice;
170
171                     oEmit =>
172                         lineNo := dummyLine;
173                         EmitOutputToken (table(tablePointer));
174                         tablePointer := tablePointer + 1;
175                         end oEmit;
176
177                     oError =>
178                         EmitError (table(tablePointer), dummyLine);
179                         tablePointer := tablePointer + 1;
180                         end oError;
181
182                     oChoiceEnd =>
183                         HandleSemanticChoiceFailure;
184                         end oChoiceEnd;
185
186
187                     { The Following are Pass dependent
188                      Data Structure Semantic Operations  }
```

```
189
190                         { No Simple, Parameterized or Choice Semantic
191                          operations in the Parser }
192
193                         oEmitIdent, oEmitChar, oEmitNumber,
194                                 oEmitString, oEmitCode, oEmitLine =>
195
196                             if semanticizing then
197                                 Semantic.EmitOp;
198                             end if;
199                             end oEmitIdent {, etc. };
200
201                     end case;
202
203                 end loop;
204
205
206                 { Terminate I/O }
207
208                 if nextInputToken not = tEndOfFile then
209                     EmitError (eExtraneousProgramText, lineNo);
210                 end if;
211
212                 IO.Close (inputFile);
213                 IO.Close (outputFile);
214                 lineNo := dummyLine;
215                 EmitError (eEndOfErrors, dummyLine);
216                 IO.Close (errorFile);
217
218
219             end;  {initially}
220
221
222     end module { Parser };
223
224 include 'IOEND'
```

A PARSER MODIFICATION OF THE EUCLID COMPILER: AUTOMATIC GENERATION OF SYNTAX ERROR RECOVERY

by

GRACE EVANS

M. S., Kansas State University, 1982

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1982

ABSTRACT

This computer project modifies the Euclid compiler's parser in order to provide; 1) more efficient syntax error recovery, 2) a syntactically correct output token stream to the next compiler pass, and 3) to demonstrate that the first two improvements can be automatically generated. Euclid, a programming language heavily indebted to Pascal, was written for the construction of verifiable systems programs, and the compiler used in this project was developed at the University of Toronto.

The parser module consists of four parts: 1) an S/SL program, 2) an S/SL assembler, 3) output from the S/SL assembler in table form, and 4) an interpreter. The S/SL (Syntax/Semantic Language--a subset of Pascal) program represents Euclid's syntax. This program is processed by an S/SL assembler module into S-code, a table of integers consisting of a few primitive operators followed by operands. The table is declared as a constant to the interpreter program which then alternately reads the token stream from the lexical analyzer and "walks" the table.

To accomplish the project's aims, several changes were required in the existing programs. First, the S-code operators were modified; then, the assembler and interpreter were changed to accomodate those modifications. In addition, syntax error recovery sets were generated by using the Hartmann error recovery scheme. The sets were computed automatically by initiating a method by which reachable tokens along a given path were stored in sets passed as operands to certain primitive operators.

The new, automatic-generation method resulting from this project insures that syntax recovery from Euclid programming errors is guaranteed and that the parser's output is syntactically correct.