Technical Report 69-85

January 1969

A Translator System
for the
EULER Programming Language

by
Victor B. Schneider

UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER
COLLEGE PARK, MARYLAND
A Translator System
for the
EULER Programming Language
by
Victor B. Schneider

The research for this paper was supported in part by N.A.S.A. grant
NsG-398 to the Computer Science Center of the University of Maryland.
Abstract

A Translator System for the EULER Programming Language.

EULER is a generalization of the ALGOL programming language, combining the ALGOL capabilities with many features similar to McCarthy's LISP language. In this paper, a new syntax is presented for EULER, together with a notation that specifies how EULER is translated. A fast, pushdown-automaton model was used as a basis for designing the translator system, and the design methods used for a portion of the translator are presented. Syntactic methods of machine-independent optimization of the translated language are discussed, and a syntactic notation for designing "extendible compilers" in terms of the EULER language is introduced.

The second section of the report is a documentation of the intermediate language into which the EULER is translated. Algorithms for executing the commands of this intermediate language are described, and versions of the programmed algorithms, rewritten in a subset of EULER, are presented.

The appendices contain listings of the actual translator and intermediate language interpreter for the EULER system. These listings consist of FORTRAN IV-MAP programs written for the IBM 7094 computer.
# TABLE OF CONTENTS

Part I. The EULER Translator .............................................. 1
Introduction .............................................................. 1
An EULER Translation Grammar .......................................... 2
Syntactic Methods for Optimizing Expressions ........................... 7
Some Syntactic Methods for Extending the EULER Language .............. 8
How A Section of the Translator Was Designed ........................... 13
Error Messages and Scans in the Translator ............................... 15
A Missing Feature of the Translator ....................................... 16
Translation Times and Memory Requirements ............................. 17

Part II. The EULER Interpretive System .................................. 18
Introduction .............................................................. 18
Structures Within the Interpreter ......................................... 19
Typing and Storage of Data ............................................... 20
Insertion of Data .......................................................... 22
The Basic Interpretive Cycle ............................................... 22
Arithmetic and Logical Operations ........................................ 23
Block Structure and Declarations ......................................... 24
Pointer Variables and Transfer of Data ................................... 27
Transfers of Control Within a Program .................................... 29
List Operations and String Processing .................................... 30
Reusage of Discarded List Cells ......................................... 33
Communication of Data .................................................... 33
Acknowledgements .......................................................... 36
Bibliography ............................................................... 37

Appendix 1. Features of the EULER Language ............................ 38
Appendix 2. SLIP Routines Used in Implementing EULER ............... 41
Appendix 3. The EULER Translator Listing ............................... 43
Appendix 4. The EULER Polish Interpreter Listing ....................... 58

ILLUSTRATION: Figure 1, A Portion of the EULER Translator .......... 14
Part I. The EULER Translator

Introduction

In a previous paper (7), we introduced methods of designing programming language translators using a pushdown automaton model and a notation for specifying how the language of the translator is mapped into its translation. In that paper, a very small programming language was used in the design of a small translator system for purposes of demonstration. This paper will present a much expanded example of translator design based on the EULER programming language of Wirth and Weber (11). Much of the discussion in this paper will draw upon results and notation used in the previous paper.

Our first order of business is a syntax of EULER. Readers of the Wirth and Weber paper will note that the original EULER grammar was written in a very stylized fashion with many extra symbols introduced in order to fit the grammar into the framework of the Wirth and Weber translation algorithm. The EULER syntax to be presented in this report has been rewritten, not for the purpose of conforming to another translation algorithm, but to show that a more nearly transparent syntax can still yield an acceptable translator. In fact, the original EULER syntax would also "work" in our system, and the new version of the EULER syntax might be thought of as an exercise in aesthetics.

During the writing of the EULER translator system, an idea began to take shape concerning simple and clever methods for extending the EULER language and for using syntactic methods for the purpose of producing partially optimized code from the translator. These methods will be discussed in the following sections that deal with optimizations and methods of extending the system.

Readers interested in learning how to use the EULER language are referred to the extremely well written presentation in the original EULER paper (11). That paper contains examples of EULER programs, a few of which have typographic errors that the careful reader can use to test his knowledge of the language. Because the EULER report deals only with the reference language (i.e., had numerous symbols not available on many card-punch machines), an appendix is included with this paper to indicate the programming conventions used with the University of Maryland EULER system. This appendix might also be a useful first guide to EULER for programmers who know some ALGOL.
An EULER Translation Grammar

The notation for the following grammar is an extension of the notation used in the previous report. In this notation, syntactic rules specifying the programming language are paired with rules of translation that specify how the language is translated, as in the following example:

<table>
<thead>
<tr>
<th>Syntactic Rule</th>
<th>Rule of Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;sum&gt; + &lt;term&gt;</code></td>
<td>I</td>
</tr>
<tr>
<td><code>+&lt;term&gt;</code></td>
<td><code>&lt;term&gt;</code></td>
</tr>
<tr>
<td><code>&lt;term&gt;</code></td>
<td><code>&lt;term&gt;.NEG.</code></td>
</tr>
<tr>
<td><code>&lt;sum&gt;+&lt;term&gt;</code></td>
<td><code>&lt;sum&gt;+&lt;term&gt;</code></td>
</tr>
<tr>
<td><code>&lt;term&gt;</code></td>
<td><code>+&lt;term&gt;</code></td>
</tr>
<tr>
<td><code>&lt;term&gt;</code></td>
<td><code>&lt;term&gt;-&lt;term&gt;</code></td>
</tr>
<tr>
<td><code>&lt;term&gt;-&lt;term&gt;</code></td>
<td><code>&lt;term&gt;-&lt;term&gt;</code></td>
</tr>
</tbody>
</table>

In this example, sequences of symbols are rearranged in the process of translation. Thus, the infix addition of "<sum>+<term>" is translated into the reverse Polish addition of "<sum><term>+". The unary plus sign in the sequence "+<term>" is discarded in the translated sequence for <term>. When the <term> stands by itself as a descendant of a <sum>, no symbols are added or deleted in the translation (hence, the notation used is I for Identity Translation).

In the rules of translation for the full grammar, there are certain sequences of symbols whose meaning may not seem obvious on a first reading. For example, to understand why the EULER statement

`A=B*C+D.IN..`,

is translated into the sequence

`.VRBL.A .VRBL.B .IN. .VRBL.C .IN.* .VRBL.D.IN..IN.+=-..`,

it is well to know something about how pointers to data are manipulated in the EULER reverse POLISH string interpreter. This interpreter is described in the next section. To understand why the EULER statement

`.IF.A.AND.B..THEN.C.ELSE.D.,`

is translated into the sequence

`.VRBL.A.IN..FAND..VRBL.B.IN..SAND..IF..VRBL.C.IN..SWCH..THEN..NOT..IF..VRBL.D.IN..SWCH..THEN...`,

it is well to read the following section on syntactic methods of producing optimized translations as well as the interpreter description. The descriptions of EULER programming in this paper and in the Wirth and Weber article will also be quite useful.
The full EULER grammar follows. A comparison of this grammar with the original one will satisfy the interested reader that the languages are virtually identical, although the grammars are different in form.

The Grammar

Syntactic Rule

<program>.ENTRY<block>.EXIT.

<block>.<blockhead><body>.END

<blockhead>.BEGIN.

|<blockhead><labeldec>.
|<blockhead><vardec>.

<vardec>.NEW.<name>

|<vardec><name>

<labeldec>.LABL.<name>

|<labeldec><name>

<body>.<body>,<stat>

|<stat>

<stat>.<labeldef><stat>

|<expr>

<labeldef>.<name>..

<expr>.<block>

|<disj>

|<var><expr>

|.GOTO.<prim>

|.OUT.<prim>

|<condition><consequence><alternative>

|<condition>.IF.<expr>

|<consequence>.THEN.<expr>

|<alternative>.ELSE.<expr>

<disj>.<conj>

|<disj>.OR.<conj>

<conj>.<neg>

|<conj>.AND.<neg>

<neg>.<relation>

Rule of Translation

<block>

I

I

I

NEW.<name>

NEW.<name>

LABL.<name>

LABL.<name>

I

I

I

.I

.I

.<prim>.GOTO.

.<prim>.OUT.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.

.<prim>.SWCH.
Syntactic Rule

| .NOT. <relation> 
<relation> =<sum> 
| <sum>(1) <relop> <sum>(2) 
<relop> = { .EQ., .NEQ., .GT., .GEQ., .LT., .LEQ. } 
<sum> = <term> 
| +<term> 
| -<term> 
| <sum>{+|-}<term> 
<term> = <factor> 
| <term>{*|/|/.|MDLO.}<factor> 
<factor> = <catena> 
| <factor>**<catena> 
<catena> = <prim> 
| <catena>.CNCT.<prim> 
<prim> = .UNDF. 
| <var> 
| <label> 
| (<expr>) 
| <procdef> 
| <referenceprim> 
| <listprim> 
| <numberprim> 
| <logicalprim> 
| .TAIL.<prim> 
| <var><list> 
| <symbolprim> 
<label> = <name> 
<var> = <name> 
| <var>.IN. 
| <var>(<sum>) 
<referenceprim> = .AT.<var> 
(listprim) = <list>

Rule of Translation

<relation> = .NOT. 
I 
<sum>(1) <sum>(2) <relop> 
I 
I 
<term> 
<term>.NEQ. 
<sum><term>{+|-} 
I 
<term><factor>{*|/|/.|MDLO.} 
I 
<factor><catena>** 
I 
<catena><prim>.CNCT. 
I 
<var>.IN. 
<label>.IN. 
<expr> 
I 
I 
I 
<prim>.TAIL. 
<var>.LSCL.<list> 
I 
<prim> .VRBL. <name> 
<var> .VRBL. <name> 
<var>.IN. 
<var><sum>) 
<var> 
I
Syntactic Rule

| .LIST.<sum> |
| .LIST.<sum> |

<list>→<listhead>.

| <listhead><expr>.
| <listhead>expr,
| <numberprim><number>.

| .REAL.<disj> |
| .LNGT.<catena> |
| .ABST.<sum> |
| .INTG.<sum> |

<logicalprim>→ .TRUE.

| .FLSE. |
| .LGCL.<sum> |

| <typeinquiry><var> |

<typeinquiry>→ { ISNU. | ISLO. | ISLA. | ISLI. | ISPR. | ISRE. | ISSY. | ISUN. }

| <symbolprim> →,* <6-symbolstring> |
| <procdef>→<prochead><expr>$.
| <prochead>$

| <prochead><formaldec>.,
| <formaldec> → .FRML.<name>
| <formaldec><name>

| <procdef> → .FRML.<name>
| <formaldec><name>

<6-symbolstring> → {<letter>|<digit>|<blank>
|.||$|*|?|=|+-|}(6
(i.e., 6 characters)

<name>→{<letter>{<letter}|<digit>}k,

k=0,1,...,5
(i.e., a name is a letter followed
by up to five letters or digits.)

<number>→<integer>

| <integer> <integer> <integer> <integer> <integer> <integer>

<integer>→<digit>

Rule of Translation

<sum>.LIST.
I
I
I
I
I
I
I
I
I
I
<sum>. ABST.
<sum>.INTG.
<sum>.LGCL.
<var><typeinquiry>
<typeinquiry>→ { ISNU. |
| ISLO. | ISLA. | ISLI. | ISPR. |
| ISRE. | ISSY. | ISUN. }

| <symbolprim> →,* <6-symbolstring>
| <procdef>→<prochead><expr>$.
| <prochead>$

| <prochead><formaldec>.,
| <formaldec><name>

| <procdef>→ .FRML.<name>
| <formaldec><name>

<6-symbolstring> → {<letter>|<digit>|<blank>
|.||$|*|?|=|+-|}(6
(i.e., 6 characters)

<name>→{<letter>{<letter}|<digit>}k,

k=0,1,...,5
(i.e., a name is a letter followed
by up to five letters or digits.)

<number>→<integer>

| <integer> <integer> <integer> <integer> <integer> <integer>

<integer>→<digit>

TEMP(1)
TEMP(1)+SCALExTEMP(2)
SCALE+0.1;
TEMP(1)+value.<digit>);
**Syntactic Rule**

|integer|digit|

<digit>→{0|1|...|9}

<letter>→{A|B|...|Z}

**Rule of Translation**

\[
\text{TEMP}(i) + 10 \times \text{TEMP}(i) + \text{value.( digit ).;}
\]

\[
\text{SCALE 0.1xSCALE;}
\]

I

I
Syntactic Methods for Optimizing Expressions

At two points in the grammar above, the rules of translation specify translated sequences of symbols that are not in reverse Polish. The motivation for doing this in the grammar is to provide a framework for executing the resulting expressions so as to skip over redundant portions of the translated string. One of the instances of this translated structure is the rule

<disj><disj>.OR.<conj>.

In ordinary Polish notation, the rule of translation would be the sequence

<disj><conj>.OR.

However, this translation does not take into account the fact that, if the <disj> is .TRUE., there is no need to evaluate the <conj> in the expression.

It was decided instead to write the rule of translation as follows:

<disj>.FSOR.<conj>.SCOR.

When this translated expression is executed, the .FSOR. operator is read immediately after the <disj> portion of the expression is evaluated. The effect of executing .FSOR. is to cause the interpreter to skip over the <conj> when the <disj> is evaluated as .TRUE.. In order to skip over the segment of program between .FSOR. and .SCOR., the interpreter routine treats the program text as though it were a table of labels, and scans ahead until it encounters an .SCOR. that matches the .FSOR. by which the routine was activated. Thus, the .SCOR. operator is not executed by the interpreter, but is used as a place marker in the translated program.

The syntactic rule <conj>-t<conj>.AND.<neg>

is matched with the translation rule

<conj>.FAND.<neg>.SAND.

for the same reason as given above. In this case, if <conj> has logical value .FLSE., the translated program segment between .FAND. and .SAND. is skipped over by the interpreter routine.

A similar rationale was used in determining the rules of translation used for conditional statements. In this case, it was decided to break down EULER conditional statements from the form

".IF.<expr>(1).THEN.<expr>(2).ELSE.<expr>(3)."

into the form

".IF.<expr>(1).THEN.<expr>(2)

.F. .NOT.<expr>(1).THEN.<expr>(3)."
In order to avoid evaluating $\text{expr}^{(1)}$ twice in the translated program, the following strategy was developed:

At run time, the EULER Polish interpreter stores the values of evaluated expressions in sequence on top of a stack of operands. It is therefore possible to retain the value of $\text{expr}^{(1)}$ for re-use in the translated program. Thus, the actual translation of the conditional statement above is the following sequence:

"$\text{expr}^{(1)}$.IF.$\text{expr}^{(2)}$.SWCH..THEN..NOT..IF.$\text{expr}^{(3)}$.SWCH..THEN..,,"

Here, the .IF. command leaves the value of $\text{expr}^{(1)}$ in place on the operand stack. If $\text{expr}^{(1)}$ is true, $\text{expr}^{(2)}$ is evaluated, and, by the EULER conventions, $\text{expr}^{(2)}$ will also reduce to a single value. When $\text{expr}^{(2)}$ is completed, the operand stack will contain the sequence value($\text{expr}^{(1)}$), value($\text{expr}^{(2)}$).

as the two topmost operands. It is then only necessary to switch these two operands (using the .SWCH. command), negate the topmost operand (using .NOT.), and skip over $\text{expr}^{(3)}$ to the .THEN. that matches the .IF. operator. The extra semicolon after the .THEN. place marker serves to unstack the value of .NOT.$\text{expr}^{(1)}$ at the end of the conditional statement. Thus, in this case also, the use of a placemarker known to the interpreter facilitates skipping over redundant portions of the translated program.

Some Syntactic Methods of Extending the EULER Language

After developing the appropriate techniques for breaking down conditional statements and for optimizing logical expressions, the next question concerns using these syntactic tricks to provide extended facilities in the EULER language. The extensions to be described in what follows have not yet been programmed into the system, there being a minor obstacle of time and money presently obstructing progress.

The introduction of full string-processing facilities into the EULER system is the first example to be considered. Without altering the EULER interpreter, and with very little reprogramming of the translator, we can effect the following
improvement:

Syntactic Rule

\[ <\text{prim}> \rightarrow <\text{stringprim}> \]
\[ <\text{stringprim}>\rightarrow <\text{stringhead}>/\]
\[ <\text{stringhead}>\rightarrow ./\]
\[ \quad | <\text{stringhead}><\text{symbol}> \]

Rule of Translation

I

Here, a string of arbitrary length is translated into a list whose cells store the symbols of the string one symbol to the cell in sequence. With this arrangement, it is possible to manipulate strings using the list concatenation operator, the \( .\text{TALI}. \) operator, and using EULER subroutines for performing tests for list equality and containment.

The second example involves the addition of facilities for reading in data at run time within the framework of the EULER system. In this case, additional structures must be provided in the EULER Polish string interpreter. These facilities would take the form of routines for converting numbers into their internal representation and for packing string data. The added syntax might resemble the following rule system:

Syntactic Rule

\[ <\text{program}>\rightarrow .\text{ENTRY}<\text{block}>.\text{EXIT}.\]
\[ \quad | .\text{ENTRY}<\text{data}>.,<\text{block}>.\text{EXIT}. \]
\[ <\text{data}>\rightarrow <\text{datahead}>.\text{END}. \]
\[ <\text{datahead}>\rightarrow .\text{DATA}..<\text{item}> \]
\[ \quad | <\text{datahead}>.,<\text{item}> \]
\[ <\text{item}>\rightarrow <\text{number}> \]
\[ \quad | <\text{stringprim}> \]
\[ \quad | <\text{data}list> \]
\[ <\text{data}list>\rightarrow <\text{data}listhead>). \]
\[ <\text{data}listhead>\rightarrow .(<\text{item}> \]
\[ \quad | <\text{data}listhead>,<\text{item}> \]

Rule of Translation

I

I

I

I

I

I

I

I

With this program structure, the \( <\text{data}> \) portion could be read in by a subroutine that leaves the \( <\text{data}> \) in a pre-arranged location in memory. The interpreter routine could then be read in on top of the data subroutine, and the translated
program would be executed. A statement of the form ".READ. prim" would then store an appropriate link to some segment of the read-in data on top of the operand stack of the interpreter.

The third example involves the use of a syntactic notation to expand the EULER Language into a self-extendible programming language similar to MAD/1 (4) and ALGOL 68 (10). By an extendible programming language, people currently mean the following two things:

(a) A language in which the programmer can specify new data types and data structures composed of novel configurations of data elements.

(b) A language in which the programmer is able to reorder the priorities of expression operators and is able to specify arbitrary new operations at will.

In EULER, there already exists a very general mechanism for allowing programmers to manipulate data structures, namely, the list mechanism. EULER lists can be constructed from arbitrary combinations of data elements. However, EULER only has eight data types with no facilities for extending their ranges. Such range-extension facilities depend on the machine on which the language is implemented, and algorithms for specifying such things as numbers of arbitrary precision must be written for the machine in question. Hence, our example will concentrate on the machine-independent problem of specifying new operators in programs.

Any reasonable programming language must presuppose the existence of a standard set of expression operators before provision is made for allowing programs to expand this set of operators. With each standard operator will be associated a standard precedence level, and the operators to be introduced by the programmer must also have precedence levels. As the term is currently used, operator precedence (or priority) is a measure of how expression operators compare in binding power. For example, exponentiation is said to have a lower precedence than addition, because exponentiation is performed before addition in arithmetic expressions. Thus, precedence imposes on the operations of a language. This ordering is reflected in the ordering of syntax rules in programming language grammars. In the EULER grammar above, rules are ordered so that list concatenation is performed first, then exponentiation, and so on, until the operation of value assignment. From concatenation to assignment of value there are nine levels of precedence.
Our approach in providing for the programming of new operations is to assign these operations to one of nine classes of operators, reflecting the nine levels in the original grammar. We accomplish this by adding an operator declaration to the language, and by permitting the programmer to define new operators and rearrange their precedences at will. The rules for permitting this in our system are as follows:

<table>
<thead>
<tr>
<th>Syntactic Rule</th>
<th>Rule of Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;expr&gt;</code>+<code>&lt;var&gt;</code>+<code>&lt;opname&gt;</code>+<code>&lt;expr&gt;</code></td>
<td><code>(.&lt;var&gt;.,$.&lt;expr&gt;$.)..CALL.&lt;opname&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>I</code></td>
</tr>
<tr>
<td></td>
<td><code>I</code></td>
</tr>
<tr>
<td></td>
<td><code>:</code></td>
</tr>
<tr>
<td></td>
<td><code>:</code></td>
</tr>
<tr>
<td><code>&lt;catena&gt;</code>+<code>&lt;catena&gt;</code>+<code>&lt;opname&gt;</code>+<code>&lt;prim&gt;</code></td>
<td><code>(.$.&lt;catena&gt;.,$.&lt;prim&gt;$.)..CALL.&lt;opname&gt;</code></td>
</tr>
</tbody>
</table>

(blockhead)→

`<blockhead>`+`<operatordec>`+`.,` `.<blockhead>`+`<operatordec>`
`<operatordec>`+.OPRT.<`opname>` `.<operatordec>`+.OPRT.<`opname>`

(Please note: Programs now have operator, data variable, and label declarations. Procedures may then have operators global to them, but may not have operators passed to them as parameters of procedure calls.)

<table>
<thead>
<tr>
<th>Syntactic Rule</th>
<th>Rule of Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;numberprim&gt;</code>+.PREC.&lt;<code>opname&gt;</code></td>
<td><code>.I</code></td>
</tr>
<tr>
<td></td>
<td><code>.I</code></td>
</tr>
<tr>
<td></td>
<td><code>.I</code></td>
</tr>
<tr>
<td></td>
<td><code>.I</code></td>
</tr>
<tr>
<td></td>
<td><code>.I</code></td>
</tr>
</tbody>
</table>

(Definition of new operators and redefinition of old ones.)
Syntactic Rule | Rule of Translation
---|---
<opdef> → <defhead><expr>$. | I
<defhead> → <formalpart><precedencepart> | I
<formalpart> → $.FRML.<name>, | $.FRML.<name>
| $.FRML.<name>,<name>, | $.FRML.<name>.FRML.<name>
<precedencepart> → RANK=<digit>. | RANK.NMBR.<digit>=.
<opname> → {|=|+-|*|..|}.CNCT. | I
| {.<letter>|<digit>}.|

In the syntax above, the <opname> in each rule is translated into a procedure call, with parameters consisting of the operands associated with each <opname>. These procedure calls either refer to the "standard" operator associated with a particular precedence level or refer to the translated <opdef>'s introduced into the program. It is assumed that the translator will automatically enclose each translated program with an extra outer block containing the procedure definitions of the standard operators. In this way, the standard operators can be redefined within a particular program, but will regain their usual meaning upon exit from the block in which the redefining statement occurred.

A certain amount of optimization is still possible within the framework of this extendible translator. As an example, suppose we write the following routine as the procedure corresponding to the standard operator for logical conjunction:

```
.AND.= $.FRML.X,Y.,RANK=7.,
.BEGIN..NEW.Z.,Z=X.,
.IF.Z.THEN.
  .IF.ISLO.Y.THEN.Y.ELSE..GOTO.ERROR
  .ELSE. Z .END.$.
```

The actual parameters in the procedure call for .AND. are subexpressions surrounded by $. Thus, the effect of the statement "Z=X."
is to evaluate the X parameter only once and not to evaluate the Y parameter unless Z is .TRUE. .
Other possible extensions to the EULER language include the addition of ALGOL-like iteration statements, as well as more rapid FORTRAN-style iterations. In both these cases, the additions can be made easily by incorporating a little extra syntax into the translator, and by having the language of that syntax translate into appropriate procedure calls to global "system" procedures.

How a Section of the Translator Was Designed—An Example

It is assumed that readers of this section will have some familiarity with the translator example in the previous paper (8) on this subject. In order to simplify the programming of the translator, it was decided to have the reserved words of the language perform as many functions as possible in the translation. Thus, the reserved words actually appear in translations as commands for the interpretive system where appropriate, and are stored on the pushdown store of the translator in place of the "nonterminal symbols" of the normal-form version of the grammar. For example, in the normal-form grammar for EULER, the rule

\[
<\text{expr}> \rightarrow X_1<\text{alternative}>
\]
\[
X_1 \rightarrow X_2<\text{consequence}>
\]
\[
X_2 \rightarrow <\text{condition}>
\]

By letting \(X_1\) be .THEN. and \(X_2\) be .IF. in the translator, the coding is greatly simplified, and no ambiguities are introduced, since the \(X_1\) can be treated as "new and distinct" symbols. The flowchart of Figure 1, showing the transitions to and from the box corresponding to <expr>, illustrates how the EULER translator given in Appendix was programmed.
Figure 1. A Portion of the EULER Translator
Error Messages and Error Scans in the Translator

In the flowchart segment of Figure 1, no error exits are indicated. This is because, with this translator-design algorithm, error exits are implied; i.e., when the appropriate combination of program and pushdown-store symbols are not read at some stage of the computation, a syntactic error in the program is indicated. Rather than halt the processing of an input program at this point, it was decided to provide a full error scan, one that points out the location of each error as it occurs.

The error-scanning mechanism is based on the following observation concerning the EULER syntax: The <body> of a program consists of sequences of the form

\[<\text{body}>.,<\text{stat}>\]

where the <stat> may be some additional sequence of statements surrounded by .BGIN. and .END.. Thus, it is possible to treat this level of the programming language as a synchronizable encoding (7) in which the decoding process can be resumed after an error occurs because the decoding algorithm does not "know" the difference between an input sequence of the form

"<\text{stat}>^{(1)}.,<\text{stat}>^{(2)}.,<\text{stat}>^{(3)}.,"

and one of the form

"<\text{stat}>^{(1)}.,<\text{stat}>^{(3)}.,"

Because of this property of the grammar, a scanning routine can be constructed so that, whenever the translator finds a program error, the routine scans forward in the program until it encounters the first semicolon or .END. following the error. This routine then resets the translator so that translation resumes at the program location following the semicolon or .END.. If a semicolon is encountered, the translator resumes translation at its initial point; if an .END. is encountered, the translator resumes translation in the section corresponding to <body>.

It is conjectured here, but not proved, that this error-scanning mechanism will never permit the printing out of a translated program containing an error. The reason for this conjecture is that the syntax-checking mechanism of the translator is critically sensitive to the balancing of .BGIN.'s and .END.'s, left-and right parentheses, etc., in a program. Thus, when an error is discovered in a program, some extra symbol will remain on the pushdown store, and this symbol will ultimately cause an error message when
the .EXIT. command of the program is read. With an error message at this point in the program, it is a very simple matter to suppress the punching out of the translated program. So far, experience with the translator has borne out this conjecture.

A Missing Feature of the Translator

In section 4.7.3.3 of the revised ALGOL report (14), the following description of procedure calls appears:

...If the procedure is called from a place outside the scope of any nonlocal quantity of the procedure body the conflicts between the identifiers inserted through this process of body replacement and the identifiers whose declarations are valid at the place of the procedure statement...will be avoided through suitable systematic changes of the latter identifiers.

The meaning of this condition can be seen in terms of the following EULER programming example:

.ENTRY.BEGIN..NEW.A,B.,
    A=2.,
    B=.$A$.,
 .BEGIN ..NEW.A.,A=5.,
 .OUT.B.END. ,
 .OUT.B .END. .EXIT.

Following the dictates of the ALGOL report, the number 2 would be written out twice by the program above. This is because the variable A is global to the definition of procedure B, and hence, some translation routine would have to rename A so that the translated program would be similar to what would appear if the above program had originally been written in the following way:

.ENTRY .BEGIN. .NEW.Z,B.,
    Z=2.,
    B=.$Z$.,
 .BEGIN ..NEW.A.,A=3.,
 .OUT.B.END. ,
 .OUT.B .END. .EXIT.

Although this ALGOL principle is not directly stated in the EULER report, there is an indication that variables in procedures are bound to the block in which they occur. Thus, something like section 4.7.3.3 must have been part of the original EULER.
In our EULER translator, the effect of executing the program example above would be to write out the number 5 followed by the number 2. Thus, section 4.7.3.3 has not yet been implemented for our system, although the programming involved is fairly straightforward. At present, the second program above must be submitted to obtain the correct results, and so the programmer must rename his program variables rather than have the translator perform the renaming.

Translation Times and Memory Requirements

The present EULER translator programmed in FORTRAN IV on the IBM 7094 machine takes approximately 5600 words of memory, including FORTRAN system routines and space for 500 packed words of EULER program to be translated. At present, 500 words of translated program output are obtained in about .06 minutes of execution time, and times for smaller programs are somewhat better, although difficult to measure.
Part II. The EULER Interpretive System

Introduction

This section consists of a description of how the EULER interpreter works. The structures used by the system and the commands that manipulate these structures are described, and algorithms written in a subset of EULER depict what happens when the commands are used. It is intended that these descriptions and algorithms, together with the actual FORTRAN IV version of the system given at the end of this report, will give the reader two somewhat different viewpoints of how the interpretive system functions.

This section of the report could also be considered an advanced manual for programming in the EULER reverse Polish language, although it is intended that the EULER translator be used for "writing" the programs used by the interpreter. It should be noted that the interpreter system now in use accepts programs written on punched cards, with the commands and data punched between columns 7 and 72 of the cards. Individual commands are punched left-justified into separate fields six columns in length. Thus, a total of eleven commands and data items can appear on one card. The EULER translator receives programs punched "free format" between columns 7 and 72 of cards and translates these programs into the interpreter notation, properly spaced and justified on punched cards.
Structures within the EULER Polish Interpreter

The following are the stacks and computer words used by the processor that executes the translated EULER code:

- **CODE** - The array in which the translated EULER program is stored.
- **I** - The index of CODE, pointing to the translated instruction currently being executed.
- **IDNTLS** - The table of names, types and values used to implement dynamic storage allocation of variables.
- **T** - The index of IDNTLS, pointing to the location of the most recently declared variable name in a program being executed.
- **OPRND** - The stack of intermediate values and links stored in the process of executing a program.
- **IOPRND** - The stack of data types associated with the values and links stored in OPRND.
- **IK** - The index of OPRND and IOPRND.
- **STORAJ** - The stack of locations in the IDNTLS name table where storage of names begins for each block.
- **BLKNUM** - The index of STORAJ that records the dynamic block number at each point in the program being executed.
- **JUMPBK** - The stack of procedure return addresses used in execution of recursive and nonrecursive procedure calls.
- **PRAMLS** - The stack of pointers to parameter lists of procedures called during program execution.
- **JMPRTN** - The index of JUMPBK and PRAMLS, giving the level of nesting of procedure calls at any point in a program.
- **LAVS** - Array name used for setting up the pool of linked list cells used by the interpreter's list-processing routines.
- **AVSL** - The machine word containing pointers to the beginning and end of LAVS.
- **LSTRTN** - A stack used for reading out the contents of lists that contain sublists during execution of a program.
- **LSTCNT** - The index of LSTRNTN, indicating the depth of the sublist currently being read using LSTRTN.
The following description of the EULER interpreter commands consists largely of algorithms written in a subset of EULER. It was felt that these algorithms can be concisely written and are intelligible to anyone familiar with ALGOL, and possibly to some familiar with MAD or FORTRAN as well. Appendix 4 is the actual interpreter program, and is written in FORTRAN IV. By comparing the EULER algorithms with the corresponding FORTRAN, it will be noted that the function JUMPTO(.*DUMMY), where .*DUMMY is some chosen operator, corresponds to several slightly different FORTRAN routines used for scanning forward in the interpreted program without executing the segment of CODE that is thereby bypassed.

**Typing and Storage of Data**

Data appears in IDNTLS, the interpreter name table, as triples of computer words. The first word in each triple is the program-declared name, the second word contains a data type code, and the third word contains the actual datum. Since the index of IDNTLS is T, this means that IDNTLS(3*T-2) is an alphanumeric word, IDNTLS(3*T-1) contains the type, and IDNTLS(3*T) contains the datum. Data is also stored on the operand stack OPRND. Here, OPRND(IK) denotes some value or linkage, and IOPRND(IK) is the integer type code of the datum in OPRND(IK).

The other important structures in which data can be stored and manipulated are list cells drawn from LAVS. Each list cell consists of a pair of adjacent computer memory words, in the manner of SLIP list cells (L1). The SLIP conventions regarding fields within a computer word are used here: That is, a given word is divided into two address-sized fields, called the LNKL and LNKR fields, and a smaller ID field. The datum is stored in the second cell of each word pair, and the typing information for that datum appears in the ID field of the first cell of the pair. To maintain a semblance of consistency within the interpreter routines, each pair of words IDNTLS(3*T-1) and IDNTLS(3*T) is made to resemble a list cell in that the typing information for the datum in IDNTLS(3*T) is stored in the ID field of IDNTLS(3*T-1). In Appendix 2 is a brief description of the SLIP routines used in this paper to describe EULER.
At present, there are eight data types manipulated by the EULER system: numerical constants and variables, logical constants and variables, labels, lists, references or pointers, procedures, alphanumeric symbols, and a last category of undefined data. The type coding mechanism works as follows:

If the ID field of IDNTLS(3*T-1) is all zero, or if IOPRND(IK) is an integer zero, the associated datum is a floating-point number. If the code is an integer one, the associated datum is logical. If the code is two, the datum is a label whose location in the CODE array is stored as a floating point number. If the ID code is three, the associated datum is an integer link to a list stored in the LNKR field of the datum word. If the ID code is four, the associated datum is called undefined. If the ID code is five, the associated datum is an integer pointer to some other datum. This link is likewise stored in the LNKR field, and is a reference to either the first word of a list cell or to IDNTLS(3*T'-1) for some T'. If the ID code is six, the datum is the location of the first command of a procedure somewhere in the CODE array, and this location is stored in floating point. Finally, if the ID code is seven, the corresponding datum contains six alphanumeric symbols, including blanks (for the IBM 7094 machine).

The following operators exist for testing the type of an operand during execution of a program. These operators check the type code of the topmost operand on the OPRND stack.

.ISNU.(isnumber) - Changes OPRND(IK) to true if IOPRND(IK) is 0. Otherwise OPRND(IK) becomes false. IOPRND(IK) becomes 1 (logical type).

.ISLO.(islogical)
.ISLA.(islabel)
.ISLI.(islist)
.ISUN.(isundefined) All defined analogously to .ISNU.
.ISRE.(isreference)
.ISPR.(isprocedure)
.ISSY.(issymbol)

The following two operators exist to perform type conversions between numbers and logical values. They permit a certain flexibility in the use of arithmetic expressions or logical expressions in "unorthodox" computations:
.LGCL.(logical type conversion)  LOGCAL:.IF.IOPRND(IK).NEQ.0.THEN.
                           .GOTO.ERROR .ELSE.IOPRND(IK)=1;.GOTO.
                           CYCLE;

.REAL.(number type conversion) TOREAL:.IF.IOPRND(IK).NEQ.1.THEN..GOTO.
                           ERROR.ELSE.IOPRND(IK)=0;.GOTO.CYCLE;

Insertion of Data

At the present time, the interpreter does not use tables of numerical and logical constants. Instead, such data is passed to the interpreter at each point of occurrence in the translated program. Numbers are passed in the form of octal representations of floating point numbers, and are stored in two consecutive six-character words. The operator .NMBR. signals to the interpreter that the following two words in CODE are a number, and a machine-language subroutine packs this number into OPRND(IK), after IK has first been incremented and IOPRND(IK) set to zero. Another operator .* signals to the interpreter that the following word in CODE consists of six alphanumeric characters. Index IK is incremented, IOPRND(IK) is set to 7, and the six-character word is stored in OPRND(IK).

Logical truth and falsity are presented to the interpreter as .TRUE. and .FALSE. respectively. For each of these operands, IK is first incremented and IOPRND(IK) set to 1. If the operand is .TRUE., OPRND(IK) is set to 1, otherwise 0. Finally, it is possible to insert .UNDF. (an undefined constant) on the operand stack. In this case also, IK is incremented. IOPRND(IK) is then set to 4, and OPRND(IK) is set to zero.

The Basic Interpretive Cycle

Several of the basic interpreter commands have already been discussed. These commands and the remaining ones to be presented in this report are all executed by a basic interpretive cycle that reads commands from the translated EULER program and passes control to routines that execute these commands. This interpretive cycle can be described in the following steps:

I. Initialization of interpreter indices:
   I=0; IK=0; BLKNUM=0; T=0; JMPRTN=0;

II. Creation of the LAVS pool of linked list cells:
    INITAS.(.AT.AVSL,.AT.LAVS,2000);;

III. Reading of the next command of the translated EULER program:
    CYCLE: I=I+1; EXECUTE.(CODE(I)); .GOTO.CYCLE;
## Arithmetic and Logical Operations in the Interpreter

The following segmented table indicates in the left column the operator in `CODE(I)` to be executed and in the right column the corresponding interpretation algorithm:

### I. Unary Operators

1. **NEG** (unary minus sign)
   - **NEGATE**: `.IF. IOPRND(IK).NEQ.0 .THEN.
   - `.GOTO. ERROR .ELSE. OPRND(IK))=-OPRND(IK);
   - `.GOTO. CYCLE;

2. **ABST.** (Take absolute value)
   - Defined analogously to unary negation.

3. **INTG.** (Truncate to an integer)

4. **NOT.** (logical negation)
   - `.NOT.:.IF. IOPRND(IK) .NEQ.1 .THEN..GOTO.
   - ERROR.ELSE.OPRND(IK)=1-OPRND(IK);
   - `.GOTO. CYCLE;

### II. Binary Operators

1. **+** (addition)
   - **ADD**: `.IF. IOPRND(IK).NEQ.0 .OR. IOPRND(IK-1)
   - .NEQ. 0 .THEN. .GOTO. ERROR .ELSE.
   - `.BGIN. OPRND(IK-1) = OPRND(IK-1)+OPRND(IK);
   - IK=IK-1; .GOTO. CYCLE .END.

2. **-** (subtraction)

3. ***** (multiplication)

4. **/** (division)

5. **/.** (integer division)

6. **%** (modulo)

7. **** (exponentiation)

   - **EQ.** (test for equality)
     - **ISSAME**: `.IF. IOPRND(IK) .NEQ. 0 .OR. IOPRND(IK-1)
     - .NEQ. 0 .THEN. .GOTO. ERROR .ELSE.
     - `.BGIN. OPRND(IK-1) = .IF. OPRND(IK-1)
     - .EQ. OPRND(IK) .THEN. 1.0.ELSE.0.0.END.
     - ;IK=IK-1; IOPRND(IK) =1; .GOTO. CYCLE;

   - **NEQ.** (test for inequality)

   - **GT.** (test for >)

   - **LT.** (test for <)

   - **GEQ.** (test for ≥)

   - **LEQ.** (test for ≤)

   - Defined analogously to the test for equality.
III. Logical Operators Modified for Increased Execution Speed

.FAND. (logical conjunction)
(Note: This is an optimized conjunction that is executed like an ALGOL conditional statement.)

.FSOR. (logical disjunction)
(Note: This is an optimized disjunction that is executed like an ALGOL conditional statement.)

.SAND. (end label of a logical conjunction)

.SCOR. (end label of a logical disjunction)

.IF. (the test operator in a conditional statement.)

.SWCH. (Exchange the two topmost operands during a conditional statement having an alternative.)

Block Structure and Declarations

Since EULER is a language having nested block structure and requiring that all program variables and labels be declared, appropriate commands exist for executing these functions. For the most part, these commands manipulate the IDNTLS table of declared names and the STORAJ stack of pointers to where the names of each currently active block are stored.
In this EULER system, the JUMPBK stack is the mechanism that implements returns of control from subroutines being executed. The stacking of subroutine return addresses permits recursive procedure calls to be executed in this system at the same speed and efficiency as non-recursive procedure calls. If a procedure leaves a value on the operand stack, that value is not destroyed by the procedure return mechanism. Thus, no extra mechanisms are necessary to implement procedures that return function values, and both types of procedures are treated alike.

There are two sequences of commands that initiate procedure calls in the EULER system. The sequence

".VRBL.<name>.IN."

where name corresponds to a procedure variable, is used for calling those procedures that have no formal parameters. The sequence

".VRBL.<name>.LSCL..(<paramlist>)."

is used for calling procedures that have formal parameters. The actual parameters in the procedure call are put onto a list, and, when the command ) is interpreted, the link to this list is put on top of the PRAMLS stack before control is transferred to that subroutine. A description of the .IN. command is given in the following section, .LSCL. is discussed in the section on transfers of control, and .( and .) are described in the section on list operations. The following list of commands and algorithms includes those commands that are interpreted when a procedure is executed by the system.
.BEGIN.
(beginning of a program block)
BEGIN:BLKNUM=BLKNUM+1;
STORAJ(BLKNUM)=T+1;
.GOTO.CYCLE:
 PROCHD:IK=IK+1;OPRND(IK)=I;
 I=JUMPTO(.*$ );
.GOTO.CYCLE;

.NEW.
(variable declaration)
NEW:I=I+1; T=T+1; IDNTLS(3*T-2)=
 CODE(I);SETDIR.(4,0,BLNKUM,
 IDNTLS(3*T-1));IDNTLS(3*T)=0;
.GOTO.CYCLE.

.LABL.
(label declaration)
LABEL:I=I+1; T=T+1; IDNTLS(3*T-2)=
 CODE(I);SETDIR.(2,0,BLNKUM,
 IDNTLS(3*T-1));:=
 SETDIR.(0,BLNKUM,LOCATE.(CODE(I)),
 IDNTLS(3*T)); .GOTO.CYCLE;

.LBDF.(label of definition)
LABDEF:I=I+1; GOTO.CYCLE;

.FRML.
(formal parameter declaration in procedure)
FORMAL:I=I+1; T=T+1; IDNTLS(3*T-2)=
 CODE(I);LINK=LNRK.(PRMMLS
 (JUMPRTN)).;IF.LINK.EQ.0.THEN.
 .BEGIN.IDNTLS(3*T)=0;
 SETDIR.(4,0,0,IDNTLS(3*T-1)).
 .END..ELSE..BEGIN.TEMP=
 ID.(INHALT.(LINK)).;IF.TEMP.EQ.
 2.OR.TEMP.EQ.5.THEN.
 SETDIR.(-1,-1,LNKL.(INHALT.(LINK+1)).
 ,IDNTLS(3*T-1))..ELSE..UNDF.;
 PRMMLS(JUMPRTN)=INHALT.(LINK);.
 IDNTLS(3*T)=INHALT.(LINK+1).
 .GOTO.CYCLE;

.. (semicolon)

.SEMCLN:.IF.IOPRND(IK).EQ.3
 .THEN..BEGIN.LNK=LNRK.(OPRND(IK)).;IF.LNKL.(INHALT.
 (LLNK).).EQ.0.THEN..ERASE.(LLNK).ELSE..UNDF..END.
 .ELSE..UNDF..IK=IK-1; GOTO.
 CYCLE;
Pointe Variables and Transfer of Data

In the EULER language, a limited use of indirect addressing exists in the form of variables of type reference. The value associated with a reference variable is either the machine address of IDNTLS(3*T'-1), for some T', or the machine address of the first word of some list cell. As an example, we could write the following fragment of a valid EULER program:

```euler
...A=2; V=.AT.A;
V.IN.=V.IN.+1....
```

Here, A is a variable of type number, and V becomes a reference to A. The last command in the fragment causes A to acquire a numerical value of 3.

On a statement-by-statement basis, the program fragment would have been translated into the following fragment: (Note that .AT. is not translated directly.)

```euler
.VRBL.A .NMBR.202400000000=
. , .VRBL.V .VRBL.A =
. , .VRBL.V .IN. .VRBL.V
.IN. .IN. .NMBR.201400000000+
```
The effect of the command .VRBL. is to fetch a pointer to the variable name that follows it in the translated program. Thus, the effect of executing the sequence

```
.VRBL.V   .VRBL.A
```

is to take the pointer for A and store it in IDNTLS as the datum for V. The effect of the command .IN. is to replace a pointer to some datum by the datum itself. Hence, the use of

```
.VRBL.V   .IN.   .IN.
```

in the translated program. Here, the datum acquired by the first use of .IN. is itself a pointer to another datum. The second use of .IN. finally captures that indirectly referenced datum. Thus, the effect of an .AT. in a program is the suppression of an .IN. in the translation.

The following list of commands presents the algorithms that are used to interpret references and assignments of value in this version of EULER. Note that error exits occur whenever some variable global to a program block is assigned as a reference to a variable local to that block.

```
.VRBL.
(fetches a pointer to a variable name)

.VRBL.
(fetches the datum referenced by a pointer; initiates procedure calls.)

.IN.
```

The following list of commands presents the algorithms that are used to interpret references and assignments of value in this version of EULER. Note that error exits occur whenever some variable global to a program block is assigned as a reference to a variable local to that block.

```
.VRBL.
(fetches a pointer to a variable name)

.VRBL.
(fetches the datum referenced by a pointer; initiates procedure calls.)

.IN.
```
Transfers of Control Within a Program

Transfers of control in a programming language are both implicit and explicit. An explicit transfer is caused by a .GOTO. command. The .LSCL. command is implicit, since it does not appear in EULER programs, but is used by the translator system for transferring control to subroutines that have parameters. Note that .LSCL. only serves to find the location in the program of the procedure. The actual transfer of control takes place when ). is read.

. GOTO.  

ERROR. ELSE. . GOTO. CYCLE. END.  
ASSIGN: IF. IOPRND(IK-1). NEQ. 5. THEN.  
.GOTO. ERROR. ELSE. . BEGIN. TYPE=IOPRND(IK);  
LRNK=LNKR. (OPRND(IK)). ; LINK=  
LNKR. (OPRND(IK-1)). ; IF. TYPE. EQ. 5. OR.  
TYPE. EQ. 3. THEN. . IF. LNKL. (OPRND(IK)).  
.GO. LNKL. (OPRND(IK-1)). . THEN. . GOTO. ERROR  
. ELSE. . UNDF. ELSE. . IF. TYPE. EQ. 3. AND.  
LRNK. GT. O. THEN. . BEGIN. SETIND. (L1,  
LNKL. (CONT. (LNK)). +1, -1, LRNK). ;  
SETDIR. (0, 0, -1, OPRND(IK)). . END. ELSE. . UNDF.  
; IF. ID. (CONT. (LINK)). EQ. 3. AND. LNKR. (  
CONT. (LINK+1)). . GO. THEN.  
ERASE. (CONT. (LINK+1)). . ELSE. . UNDF.;  
SETIND. (TYPE, -1, -1, LINK). ; OPRND(IK)=  
CONT(LINK+1); OPRND(IK-1)=OPRND(IK);  
IOPRND(IK-1)=IOPRND(IK); IK=IK-1; . GOTO. CYCLE;

JUMP: IF. IOPRND(IK). NEQ. 2. THEN.  
.GOTO. ERROR. ELSE. . BEGIN. I=LNKR. (  
OPRND(IK)). -1; JMPRTN=JMPRTN  
+LNKL. (OPRND(IK)). -BLKNUM; . IF.  
JMPRTN. LT. O. THEN. JMPRTN=O. ELSE.  
.UNDF.; . GOTO. CYCLE. END.
List Operations and String Processing

The EULER system has facilities for constructing and concatenating lists, for taking the suffix of a list, and for finding subscripted list elements. Moreover, each list call can store data of any legal type, including procedure, reference, label, etc. Thus, it is possible to execute a program by simply naming the elements of a list of procedures in order. Or, we can obtain the effect of an ALGOL switch by constructing a list of labels and writing .GOTO. followed by a subscripted list reference. Since characters can be stored on lists, it is possible to compare long character strings by writing procedures for testing two lists for equality, containment, etc.

The command .LIST. constructs an empty list whose length is given by the numerical value of the topmost operand on the OPRND stack. The commands .( and ). construct a list out of data elements that occur between them separated by commas. The subscripting command ) is evaluated one subscript level at a time and takes its information from the two topmost elements of the OPRND stack. The concatenation operator .CNCT. creates a copy of the two lists whose pointers are the topmost operands of the OPRND stack. Finally, the suffix operator .TAIL. finds the link to a list on top of the OPRND stack and replaces that link by a link to the following cell of that list.

The following example illustrates a distinction that should be understood when programming EULER: If A and B are lists in a program, then the two statements

\[ C = (A, B) \text{; and } C = A \text{CNCT} B; \]

do not have the same effect. In the first case, C is a list of two elements which point to A and B as sublists. In the second case, C is a list consisting of a copy of A joined to a copy of B.

The following table of commands and algorithms further describes the results produced by the list manipulation operators:
.LIST.

(stores topmost OPRND in a list cell and replaces this operand with a link to the cell.)

LIST:.IF.IOPRND(IK).NE.O.THE.N.

..GOTO.ERROR.ELSE..BGIN.IOPRND(IK)

=3;INDEX=OPRND(IK);.IF.INDEX

..EQ.0.THE.N..BGIN.OPRND(IK)=0;

..GOTO.CYCLE.END..ELSE..UNDF.;

IVRBL=NUCELL;.IF.INDEX

..EQ.1.THE.N..GOTO.OUTPUT.ELSE.LLCELL=

IVRBL;LOOP2:INDEX=INDEX-1;.IF.

INDEX.LT.1.THE.N..GOTO.OUTPUT.ELSE.

..BGIN.LRCELL=NUCELL;SETIND.(4,

-1,LRCELL,LLCELL);LLCELL=LRCELL;

..GOTO.LOOP2.END.;OUTPUT:SETDIR.(0,

0,IVRBL,OPRND(IK));

..(.

LISTHD:IK=IK+1;IOPRND(IK)=3;

OPRND(IK)=.*( ;GOTO.CYCLE;

COMMA:IKK=IOPRND(IK);IJJ=LNKR.(.

OPRND(IK)).;

..IF.IKK.EQ.3.AND.IJJ.GT.0.THE.N.

SETIND.(-1,LNKL.(CONT.(IJJ)).+1,-1,IJJ).

..ELSE..UNDF.;TEMP=NUCELL;

SETIND.(IKK,0,0,ITEMP);STRIND.(.

OPRND(IK),ITEMP+1);OPRND(IK)=TEMP;

IOPRND(IK)=3;GOTO.CYCLE;

LSTEND:.IF.OPRND(IK).EQ..*( .THEN.

..BGIN.OPRND(IK)=0;GOTO.CYCLE.END.

..ELSE.EXECUTE.(COMMA);LOOP3:.IF.

OPRND(IK-1).NEQ..*( .THEN..BGIN.

SETIND.(-1,0,LNKR.(OPRND(IK)),LNKR

.(OPRND(IK-1)).;IK=IK-1;GOTO.LOOP3

..END..ELSE..IF.OPRND(IK-1).NEQ.6.THE.N.

..GOTO.CYCLE.END..BEGIN.JMPRTN=1+

JMPRTN;JUMPBK(JMPRTN)=I;T=T+1;

PRAMLS (JMPRTN)=IDNTLS(3*T)=OPRND(IK);

SETDIR.(3,1,0,IDNTLS(3*T-1));I=OPRND(IK-1;

IK=IK-2;GOTO.BEGIN;
RSBSCP: INDEX=OPRND(IK); IK=IK-1;

IF INDEX.LEQ.0 THEN GOTO CYCLE
ELSE LINK=LNKR.(CONT.(1+LNKR.(OPRND(IK)).)); IF LINK.GT.0 THEN.
BEGIN LOOP4: INDEX=INDEX-1; IF INDEX
GT.0 THEN BEGIN LINK=LNKR.(CONT.
LINK).); GOTO LOOP4. END ELSE BEGIN.
SETDIR.(-1,-1,LINK,OPRND(IK)); GOTO CYCLE
END END ELSE GOTO ERROR;

TAIL:

TAIL: TEMP=LNKR.(OPRND(IK)); IF.
IOPRND(IK).NE.3 OR TEMP.EQ.0 THEN.
GOTO ERROR ELSE TEMP=LNKR.(CONT
(TEMP).); IF TEMP.EQ.0 THEN GOTO.
ERROR ELSE BEGIN OPRND(IK)=TEMP;
SETIND.(-1,-1,LNKR.(CONT.(TEMP).),-1,TEMP).
GOTO CYCLE END;

CNCT:

CONCAT: IF IOPRND(IK).NE.3 OR.
IOPRND(IK-1).NE.3 THEN GOTO.
ERROR ELSE BEGIN COPY.(LNKR.(OPRND(IK)).,
JTOP,JBOT); COPY.(LNKR.(OPRND(IK-1)).,
ITOP,IBOT); SETIND.(-1,-1,ITOP,IBOT);
IK=IK-1; OPRND(IK)=ITOP; GOTO CYCLE
END.

LNCT:

LIST: IF IOPRND(IK).NE.3
THEN GOTO ERROR ELSE BEGIN COUNT=0;
TEMP=LNKR.(OPRND(IK)); IOPRND(IK)=0;
LOOPS: IF TEMP.EQ.0 THEN BEGIN OPRND(IK)
=COUNT; GOTO CYCLE END ELSE BEGIN.
COUNT=1+COUNT; TEMP=LNKR.(CONT
(TEMP).); GOTO LOOPS END.
Reusage of Discarded List Cells by the System

The EULER programmer will not usually concern himself with the problem of "garbage collection" of discarded list cells in his programs, since garbage collection is automatic in EULER. This collection occurs at three points in EULER programs: at the .END. command of a block, at an assignment statement when the = command is executed, and between statements when the ; command is executed. At the end of a block, all lists local to the block are linked one after another to the last cell in the LAVS list. When an assignment statement assigns a new value to a variable that was formerly a list, that list is checked to see if its first cell only has one name. If that list only has one name, it is linked to the last cell of LAVS. When a semicolon is encountered in the translated program, the top of the OPRND stack is checked to see if it contains a link to some list having no name (such as would be left there if something like .OUT..( ). were the preceding statement). If there is a link to an anonymous list on the OPRND stack, then this link is removed from OPRND, and the list is joined to the end of LAVS.

As in the SLIP system (a part of which is used in our implementation of EULER), sublists of lists are joined to the end of LAVS only when the NUCELL routine encounters them in the process of fetching list cells from the top of LAVS. In this case, the sublists are only considered to be reusable if they are sublists of no more than one list (in this case, they are sublists of LAVS). More information on this garbage collection technique can be found in the original SLIP paper. (11)

Communication of Data

The EULER language allows simple variables and lists to be printed out at execution time. A program statement such as

.OUT.V;

is translated into the sequence

.VRBL.V IN. .OUT. ;

If V is a list, then the entire list, including sublists will be printed out. If V is a variable, then just that variable's value will be printed out. If V is a procedure, reference, or label, then PROCDR, REFRNC, or LABEL will be the appropriate "values" to be printed by the .OUT. operator.
As a further example, we might have the following statement in an EULER program:

```
.OUT. (.N= ,3.2,.THIS I,.S A LI,.ST WIT,
.*H SUBL,.INSTS .,.(.$3$,.AT.N,.(2,3).).);
```

When the corresponding statement in the translated program is executed, the following printed output will result:

```
( N= 3.2000,THIS IS A LI
ST WITH SUBLISTS. , ( PROCDR,REFRNC,
( 2.0000, 3.0000 ) )
```

Note that commas are suppressed between list elements containing characters.

A discussion of how one might extend the printing facilities of EULER can be found in the section entitled "Improvements and Extensions of EULER".

There are no facilities for reading data in the EULER language. The original EULER paper suggests that data be translated with the program, since translation can be a very rapid process. Perhaps a better suggestion, and one that we hope to implement, is to allow a list-structured input of data similar to the one used with the .OUT. command. The "Improvements and Extensions" Section of this paper has more to say on this matter.

The following algorithm implements list and variable printing as described above. The OUTPUT subroutine prints one datum for each call, and an index is used by the subroutine to position the datum on a printed line. Note that the LSTRTN stack facilitates writing out of sublists.

```
.OUT. OUT: TYPE=IOPRND(IK);.IF.TYPE.NEQ.3
.THEN..BGIN.INDEX=1; OUTPUT.(TYPE,OPRND(IK),
.AT.INDEX).;.GOTO.CYCLE.END..ELSE..IF.LNKR.(OPRND(IK))..EQ.0..THEN..BGIN.INDEX=1;
OUTPUT.(7,* ( ),.AT.INDEX;.GOTO.CYCLE.END.
.ELSE..BGIN.
LSTCNT=INDEX=0;OUTPUT.(7,* ( ,.AT.INDEX).
;LINK=LNKR.(OPRND(IK));LOOP6:CTLNIK=CONT.(LINK);.IF.ID.(CTLNIK).EQ.3..THEN..BGIN.
LSTCNT=1+LSTCNT;LSTRTN(LSTCNT)=LINK;
LINK=LNKR.(CONT.(LINK+1));OUTPUT.(7,* ( ,
.AT.INDEX).;.GOTO.LOOP6.END.
```
.ELSE..BEGIN_OUTPUT. ID.(CTLINK), CONT.(LINK+1),
.AT_INDEX); IF .ID.(CTLINK) .NEQ. 7 AND
LNKR.(CTLINK) .GT. 0 THEN OUTPUT.(7, ,
.AT_INDEX); ELSE UNDF END;
ENDIF.(CTLINK) .GT. 0 THEN BEGIN LINK=
LNKR.(CTLINK); GOTO LOOP6 END ELSE.
BEGIN LOOP7: OUTPUT.(7,* ) , AT_INDEX); IF
LSTCNT.EQ.0 THEN GOTO CYCLE
ELSE BEGIN LINK=LNKR.(CONT.(LSTRTN(LSTCNT)));
;LSTCNT=LSTCNT-1; IF LINK.EQ.0 THEN.
GOTO LOOP7 ELSE BEGIN OUTPUT.(7,* , , AT.
INDEX); GOTO LOOP6 END END.

END.
Acknowledgements

During the programming of the EULER reverse Polish interpreter, Charles Mesztenyi and John Pfaltz of the University of Maryland Computer Science Center very kindly offered suggestions on programming techniques that materially hastened the completion of the interpreter. The author also wishes to acknowledge discussions on the subject of the translator with John C. Reynolds of the Argonne National Laboratory and Martin Milgram of the University of Maryland. Daniel Fishman and Frank Adair of the Computer Center also gave of their time in the early programming of the interpreter.
Bibliography


Appendix 1. Features of the EULER Language

EULER is a nested block-structure language, similar to ALGOL. Thus, every block, consisting of a sequence of statements surrounded by .BEGIN. and .END. parentheses, can be treated as a single statement in ALGOL fashion. An EULER program consists of an EULER block preceded by .ENTRY and followed by .EXIT.

In EULER, there are three declarations, one category for data variables, one for program labels, and one for formal parameters of procedures. In the program

".ENTRY .BEGIN .NEW.X,Y,.LABL.Z,. Z..X=Y .END. .EXIT."

X and Y will store data, and Z will be a label preceding some statement.

Assigning a data type to a declared variable is accomplished by writing an assignment statement with data of the appropriate type on the right-hand side of the assignment. Thus, typing of variables in EULER is dynamic, since any assignment statement can change the data type stored in a variable. And, data typing is implicit, since there are no declarations like real, integer, etc., as appear in ALGOL. The following is a list of the eight EULER data types:

I. Number - In the EULER system, all numbers are assumed to be floating point numbers. The assignment statement

"V=E."

with E a numerical expression or number, causes variable V to become a numerical variable.

II. Symbol - In this EULER implementation, an assignment statement such as

"V=.*ALPHAN."

causes the six characters "ALPHAN" to be stored in the location named by variable V.

III. Logical - The logical constants are .TRUE. and .FLSE., standing respectively for logical truth and falsehood. The assignment statement,

"V=L."

with L a logical constant or logical expression, causes variable V to become a logical variable.
IV. Label - It will be recalled that EULER programs declare .NEW. variables as well as .LABL. statement labels. Interestingly enough, if \( V \) is a variable in some EULER program, and \( V \) is not in a block global to the block of label \( L \), then the assignment statement 

\[
V = L, 
\]

causes \( V \) henceforth to be of type label, and to be interchangeable with \( L \) in .GOTO. statements.

V. Reference - In EULER, if \( V_1 \) is a variable not in a block global to the block of variable \( V_2 \), then the assignment statement 

\[
V_1 = \text{AT}. V_2, 
\]

makes \( V_1 \) a pointer to the data stored in \( V_2 \). After \( V_1 \) is turned into such a pointer, the two statements 

\[
V_2 = V_2 + 1, 
\]

and 

\[
V_1.\text{IN.} = V_1.\text{IN.} + 1, 
\]

will have exactly the same effect of manipulating whatever data is stored in \( V_2 \).

VI. Procedure - An assignment statement of the form 

\[
V_1 = \text{$.}\text{<expr>$.}, 
\]

causes \( V_1 \) to become the name of a parameterless procedure call with body given by \( \text{<expr>} \). As a programming example, we might consider the following EULER block:

\[
\text{.BEGIN.}. \text{NEW.X,Y ., X=2;}
\]

\[
Y = \text{$.}\text{FRML.Z. , X=X+Z$.}, 
\]

\[
\text{.OUT.Y.(5).=.END.} 
\]

When \( Y.(5). \) is operated on by the .OUT. operator, the value 7.0000 will be written out.

VII. List - In EULER, lists can be constructed in three distinct ways:

(a) On command: "\( V_1 = \text{.LIST.300.} \)"

This statement creates a list of 300 cells and makes \( V_1 \) their name.
(b) By explicit notation: "\( V2 = (1, (2, 3), 4) \)"
This statement creates a list consisting of two numbers and a sublist and makes \( V2 \) the name of that list.

(c) By concatenation: "\( V1 = V1 \cdot \text{CNCT} \cdot V2 \)"
Using the concatenation operator, small lists can be joined together into larger ones. In addition, lists can be subscripted in the same way as ALGOL arrays, and each element of a list can be any EULER data type, including label, reference, and procedure. The following EULER block is an example of the generality of the list notation:

\begin{verbatim}
.BEGIN..NEW.X,Y.,.LABL.Z.,
Y=(2,.$BGIN.X=X+1.,Y(X).END.$.,
$.OUT.X$.,$Z$.,
X=Y(1),Y(X),.GOTO.Y(4),
Z..OUT..*FINISH.END.
\end{verbatim}

With this program segment, first 3.0000, then FINISH, will be written out by the executed program.

VIII. Undefined - Every variable declared by .NEW. in an EULER program is initially of type .UNDF.(undefined). In addition .UNDF. is used as a data constant occasionally and as an empty option in conditional statements, such as:

"\( V = \text{IF}.L1.\text{THEN}.(1,5).\text{ELSE}.UNDF. \)"

For more details on EULER programming the reader is referred to the Wirth and Weber EULER paper (11).
Appendix 2. The SLIP Routines Used in Implementing EULER

The current EULER implementation uses lists whose cells consist of pairs of memory words having the following format:

Machine Address = M
Machine Address = M + 1

<table>
<thead>
<tr>
<th>Type</th>
<th>LNKL</th>
<th>LNKR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Datum</td>
<td></td>
</tr>
</tbody>
</table>

The LNKR field of word M stores the machine address of the first word of the next cell in a list. If LNKR = 0, then that cell is the last cell of some list. The LNKL field stores a count of how many program names refer to a cell; i.e., the LNKL measures how many times a programmer has made that cell the head of a list. The type field, of course, stores the type code of the datum or link contained in word M + 1.

In the EULER algorithms describing the commands of the reverse Polish interpreter system, the following subroutines are used for list manipulation:

I. INITAS.(.AT.AVSL,.AT.LAVS,2000).

This routine passes pointers from the word AVSL and the 200 word array LAVS to a subroutine that organizes LAVS into a pool of list cells. A pointer to the first cell of this pool is stored in the LNKR field of AVSL, and the LNKL field of AVSL stores a pointer to the last cell of the LAVS pool.

II. ID.(WORD), LNKL.(WORD), LNKR.(WORD).

These functions perform appropriate masking and shifting of the contents of the <name> WORD, and return as values the ID, LNKL, and LNKR fields of WORD, respectively.

III. CONT.(ADDRESS).

This function returns the datum stored in the memory word with location given by ADDRESS.
IV. SETDIR. (ID, LNKL, LNKR, NAME).
    Packs the value of ID into the ID field, LNKL into the LNKL field, and LNKR into the LNKR field of the word NAME. If any of the first three parameters of SETDIR is -1, the corresponding field of NAME is left unchanged.

V. SETIND. (ID, LNKL, LNKR, ADDRESS).
    Similar to SETDIR above, except that ADDRESS is the machine address of some word in memory.

VI. MADOV. (NAME).
    This function returns the machine address of the word containing the data stored in NAME.

VII. NUCELL
    This parameterless function returns a link to the first free list cell in the LAVS pool. NUCELL also resets the LNKR field of the AVSL word to point to the new head of LAVS, and NUCELL initiates garbage collection of any unused sublists that it encounters.
Appendix 3. The EULER Translator Listing

C NAMING AND TYPING OF STRUCTURES IN THE EULER TRANSLATOR.
LOGICAL UNSAME,NOTIUT
DIMENSION IDNTLS(1000),INPUT(3000),OUTPUT(1000),N(3000),
STORAJ(200)
EQUIVALENCE (INPUT(501),OUTPUT(1)),(INPUT(1501),IDNTLS(1)),
(OUTPUT(2501),N(1)), (INPUT(2801),STORAJ(1))
INTEGER OUTPUT,PROCID,PRCEND,BEGIN,TI.MNAT,SEMCLN,OLON,ASSINE,
OUT,TEST,SWICH,THEN,ELSE,OR,FRSTOR,SCNDOR,AND,FSTAND,SCNAND,
GREAR,GREATQ,PLUS,ADD,SBTLC,FLTDIV,EXPNT,CONCAT,UNDFDN,
RTPREN,TAIL,FORMAL,COMMA,VARIBL,AT,TOREAL,TOABS,TOINT,TRUTH,
FALSY,SYMBO,BLKNUM,STORAJ
DATA KBGIN,KEND/6H,ENTRY/6H,EXIT/6H,ENTRY/6H,EXIT/
DATA ISBLAN,NUMBER,LENGTH,LOGCAL/
1 1H0,6.1,NMFR,6H,LMGT,6H,LCMc/
DATA ISNMFR,ILGCL,ISLAB,ISLIST,ISUNDF,ISRFN,ISPRCD,ISMAME/
1 6H,ISNU,6H,ISLO,6H,ISLA,6H,ISLI,6H,ISL,6H,ISU,6H,ISRE,6H,
2 6H,ISPR,6H,ISSY/
DATA NEWS,LABEL,JUMP,NEGATE,LABDEF,ISSAME,NOTSAM/
1 5H,NEW,6H,LABL,6H,CHOTO,5H,NOT,6H,LCDF,4H,EQ,5H,NEQ,/
DATA LESSN,LESSQ,NEG,MO,DUO,MTPLY,INTDIV/
1 4H,LT,5H,EQ,5H,NEG,6H,MLE,1H%/3H%/3H/
DATA INSIDE,LFPREN,LSCALL,LIST,LISTD,LST,END,LSTSC/P
1 4H,IN,1H6,H,LSCL,6H,LIST,1H2,H,2H,4H,1H%/1H%
DATA PROCID,PROCEND,BEG,MTNAT,SEMCLN,OLON/
1 2H,SY,2H,5H,BGIN,5H,END,2H,5H,1H%/3H%
DATA ASSINE,OUT,TEST,SWICH,THEN/
1 1H,5H,UT,4H,IF,6H,SWCH,6H,THEN/
DATA GREATER,GREATQ,PPLUS,ADD,SBTLC,FLTDIV,EXPNNT/
1 4H,GT,5H,EQ,6H,PLUS,1H,1H,1H,2H%/2H%
DATA ELSE,OR,FRSTOR,SCNDOR,AND,FSTAND,SCNAND/
1 6H,ELSE,4H,QR,6H,FSOR,6H,SCOR,5H,AND,6H,FAND,6H,SNAD/
DATA CONCAT,UNDFDN,RTPREN,TAIL,FORMAL,COMMA,VARIBL/
1 6H,CNCT,6H,UNDF,1H,6H,TAIL,6H,FRML,1H,6H,VRBL/
DATA AT,TOREAL,TOABS,TOINT,TRUTH,FALSY,SYMBO/
1 4H,AT,6H,REAL,6H,ABST,6H,INTG,6H,TRUE,6H,FLSE,2H%/2H%
C C INITIALIZATION--IT INDEXES IDNTLS, JK INDEXES INPUT, LL INDEXES OUTPUT
C AND IK INDEXES N, BLKNUM INDEXES STORAJ, MARK IS USED IN WRITING OUT
C TRACES FOR BAD PROGRAMS.
C
IT=0
JK=1
LL=0
IK=0
BLKNUM=0
MARK=1
CALL EDITOR(INPUT)
WRITE(6,4)
FORMAT(1H1)
C INITIAL POINT FOR TRANSLATION
C
IF(UINSAME(INPUT(JK),KBGIN)) GO TO 8000
CALL STORE(KBGIN,N,IK,JK)
C INITIAL POINT FOR BLOCK.
100  ICODE=INPUT(JK)
1  IF(UNSAME(ICODE,ISNAME) .AND. UNSAME(ICODE,ISLGC)) GO TO 110
2  UNSAME(ICODE,ISLIST) .AND. UNSAME(ICODE,ISFRN) .AND.
3  UNSAME(ICODE,ISPRCD) .AND. UNSAME(ICODE,ISSYM)) GO TO 110
4  CALL STORE(ICODE,N,IK,JK)
5  GO TO 200

C  200  IS INITIAL POINT OF PRIMARY.

110  IF(UNSAME(ICODE,LSTHD)) GO TO 120
1  CALL OUTCOD(LSTHD,OUTPUT,LL)
2  CALL STORE(LSTHD,N,IK,JK)
3  GO TO 100

120  IF(UNSAME(ICODE,BEGIN) .AND. UNSAME(ICODE,PROCID)) GO TO 130
1  BLKNUM=BLKNUM+1
2  CALL OUTCOD(ICODE,OUTPUT,LL)
3  CALL STORE(ICODE,N,IK,JK)
4  GO TO 100

130  IF(UNSAME(ICODE,ADD)) GO TO 140
1  CALL STORE(PLUS,N,IK,JK)
2  GO TO 100

140  IF(UNSAME(ICODE,SUBTRC)) GO TO 150
1  CALL STORE(NEG,N,IK,JK)
2  GO TO 100

150  IF(UNSAME(ICODE,TAIL) .AND. UNSAME(ICODE,TOUT) .AND. UNSAME(ICODE,
     NOT) .AND. UNSAME(ICODE,TEST) .AND. UNSAME(ICODE,THEN) .AND.
1  LOGCAL) .AND. UNSAME(ICODE,TOREAL) .AND. UNSAME(ICODE,LENGTH)
2  .AND. UNSAME(ICODE,AT) .AND. UNSAME(ICODE,TOINTG) .AND. UNSAME(
3  ICODE,TOASL) .AND. UNSAME(ICODE,LIST) .AND. UNSAME(ICODE,Jump))
4  GO TO 160
5  CALL STORE(ICODE,N,IK,JK)
6  GO TO 100

160  IF(UNSAME(ICODE,NEW) .AND. UNSAME(ICODE,FORMAL) .AND.
1  UNSAME(ICODE,LABEL)) GO TO 200
2  CALL STORE(ICODE,N,IK,JK)

C  C  INITIAL POINT FOR PRIMARY.
C
C  200  ICODE=INPUT(JK)
1  IF(UNSAME(ICODE,SYMBOL)) GO TO 210
2  CALL OUTCOD(SYMBOL,OUTPUT,LL)
3  JK=JK+1
4  CALL OUTCOD(INPUT(JK),OUTPUT,LL)
5  JK=JK+1
6  GO TO 300

210  IF(UNSAME(ICODE,NUMBER)) GO TO 220
1  CALL OUTCOD(NUMBER,OUTPUT,LL)
2  CALL OUTCOD(INPUT(JK+1),OUTPUT,LL)
3  CALL OUTCOD(INPUT(JK+2),OUTPUT,LL)
4  JK=JK+3
5  GO TO 300

220  IF(UNSAME(ICODE,TRUTH) .AND. UNSAME(ICODE,FALSTY) .AND. UNSAME(
1  ICODE,UNDFN)) GO TO 230
2  CALL OUTCOD(ICODE,OUTPUT,LL)
3  JK=JK+1
4  GO TO 300

230  IF(NOT(1D(ICODE)) GO TO 8230
1  IF(UNSAME(INPUT(JK+1),COLON)) GO TO 235
2  CALL OUTCOD(LABDEF,OUTPUT,LL)
3  CALL OUTCOD(ICODE,OUTPUT,LL)
JK = JK + 2
GO TO 100

235  JK = JK + 1
NOFIK = N(IK)
IF(UNSAME(NOFIK, FORMAL)) GO TO 250
ITEMP = FORMAL
CALL OUTCOD(FORMAL, OUTPUT, LL)
CALL OUTCOD(ICODE, OUTPUT, LL)
IT = IT + 1
IDNTLS(IT) = ICODE
IK = IK - 1
ICODE = INPUT(JK)
IF(UNSAME(ICODE, COMMA)) GO TO 240
CALL STORE(ITEMP, N, IK, JK)
GO TO 200

240 IF(UNSAME(ICODE, SEMCLN)) GO TO 2440
JK = JK + 1
IF(UNSAME(N(IK), PROCHD)) GO TO 245
GO TO 100

250 IF(UNSAME(NOFIK, NEW) AND UNSAME(NOFIK, LABEL)) GO TO 260
ITEMP = NOFIK
CALL OUTCOD(ITEMP, OUTPUT, LL)
CALL OUTCOD(ICODE, OUTPUT, LL)
IT = IT + 1
IDNTLS(IT) = ICODE
IK = IK - 1
ICODE = INPUT(JK)
IF(UNSAME(ICODE, COMMA)) GO TO 255
CALL STORE(ITEMP, N, IK, JK)
GO TO 200

255 IF(UNSAME(ICODE, SEMCLN)) GO TO 2555
JK = JK + 1
IF(UNSAME(N(IK), BEGIN)) GO TO 256
GO TO 100

C ICODE IS STILL THE IDENTIFIER IN QUESTION.

260 ITEM = IT
261 IF(NOT UNSAME(IDNTLS(ITEMP), ICODE)) GO TO 265
ITEMP = ITEMP - 1
IF(ITEMP) 8261, 8261, 261
265 CALL OUTCOD(VARIBL, OUTPUT, LL)
CALL OUTCOD(ICODE, OUTPUT, LL)

C 266 IS THE LABEL VARIABLE IN THE FLOWCHART.

C JK WAS INCREMENTED AT LABEL 235. ICODE WAS THEN INPUT(JK-1).

266 ICODE = INPUT(JK)
IF(UNSAME(ICODE, INSIDE)) GO TO 269
CALL OUTCOD(INSIDE, OUTPUT, LL)
JK = JK + 1
GO TO 266

269 IF(UNSAME(ICODE, ASSINE)) GO TO 275
270 CALL STORE(ICODE, N, IK, JK)
GO TO 100

275 IF(UNSAME(ICODE, LISTHD)) GO TO 277
277 CALL OUTCOD(LSCALL, OUTPUT, LL)
CALL OUTCOD(LISTHD, OUTPUT, LL)
GO TO 270

277 IF(UNSAME(ICODE, LFPREN)) GO TO 280
280 NOFIK = N(IK)
IF(UNSAME(NOFIK, AT)) GO TO 285
IK=IK-1
GO TO 300

285 IF(UNSAME(NOFIK, ISNUMB) AND UNSAME(NOFIK, ISLGCL) AND
1 UNSAME(NOFIK, ISLACL) AND UNSAME(NOFIK, ISLIST) AND
2 UNSAME(NOFIK, ISUNDF) AND UNSAME(NOFIK, ISRFRN) AND
3 UNSAME(NOFIK, ISPRCD) AND UNSAME(NOFIK, ISSYM)) GO TO 295
CALL OUTCOD(NOFIK, OUTPUT, LL)
IK=IK-1
GO TO 300

295 CALL OUTCOD(INSIDE, OUTPUT, LL)

C
300 IS THE LABEL OF PRIMARY.

C
300
NOFIK=N(IK)
IF(UNSAME(NOFIK, TAIL)) GO TO 305
CALL OUTCOD(NOFIK, OUTPUT, LL)
IK=IK-1
GO TO 300

305 IF(UNSAME(NOFIK, JUMP) AND UNSAME(NOFIK, OUT)) GO TO 310
CALL OUTCOD(NOFIK, OUTPUT, LL)
IK=IK-1
GO TO 1000

C
1000 IS THE LABEL OF EXPRESSION.

C
310 IF(UNSAME(NOFIK, CONCAT)) GO TO 400

C
400 IS THE LABEL OF CATENA.
CALL OUTCOD(NOFIK, OUTPUT, LL)
IK=IK-1

C
400 IS THE LABEL OF CATENA.

C
400
IF(UNSAME(INPUT(JK), CONCAT)) GO TO 410
CALL STORE(CONCAT, N, IK, JK)
GO TO 100

410 NOFIK=N(IK)
IF(UNSAME(NOFIK, LENGTH)) GO TO 420
CALL OUTCOD(LENGTH, OUTPUT, LL)
IK=IK-1
GO TO 300

420 IF(UNSAME(NOFIK, EXPNNT)) GO TO 500
CALL OUTCOD(EXPNNT, OUTPUT, LL)
IK=IK-1

C
500 IS THE LABEL OF FACTOR.

C
500
IF(UNSAME(INPUT(JK), EXPNNT)) GO TO 520
CALL STORE(EXPNNT, N, IK, JK)
GO TO 100

520 NOFIK=N(IK)
IF(UNSAME(NOFIK, MLTPLY) AND UNSAME(NOFIK, INTDIV) AND
1 UNSAME(NOFIK, FLTDIV) AND UNSAME(NOFIK, MODULO)) GO TO 600
CALL OUTCOD(NOFIK, OUTPUT, LL)
IK=IK-1

C
600 IS THE LABEL OF TERM.

C
600
ICODE=INPUT(JK)
IF(UNSAME(ICODE, MLTPLY) AND UNSAME(ICODE, INTDIV) AND
1 UNSAME(ICODE, FLTDIV) AND UNSAME(ICODE, MODULO)) GO TO 610
CALL STORE(ICODE, N, IK, JK)
GO TO 100
47

610   NOFIK=N(IK)
       IF(UNSAME(NOFIG,PLUS).AND.UNSAME(NOFIG,NEG).AND.UNSAME(NOFIG,
       1       ADD).AND.UNSAME(NOFIG,SUBIRC)) GO TO 700
       CALL OUTCOD(NOFIG,OUTPUT,LL)
       IK=IK-1

C     700 IS THE LABEL OF SUM.
C
C
C 700   ICODE=INPUT(JK)
       IF(UNSAME(ICODE,ISSAME).AND.UNSAME(ICODE,NOTSAM).AND.
       1       UNSAME(ICODE,LESSN).AND.UNSAME(ICODE,LESSEQ).AND.
       2       UNSAME(ICODE,GREATR).AND.UNSAME(ICODE,IGRATEQ).AND.
       3       UNSAME(ICODE,ADD).AND.UNSAME(ICODE,SUBIRC)) GO TO 710
       CALL STORE(ICODE,N,IK,JK)
       CALL OUTCOD(ICODE,N,OUTPUT,LL)
       IK=IK-1
       GO TO 100

710   NOFIK=N(IK)
       IF(UNSAME(NOFIG,TOING).AND.UNSAME(NOFIG,TOABS).AND.
       1       UNSAME(NOFIG,LOGCAL).AND.UNSAME(NOFIG,LIST)))GO TO 720
       CALL OUTCOD(NOFIG,OUTPUT,LL)
       IK=IK-1
       GO TO 300

720   IF(UNSAME(NOFIG,ISSAME).AND.UNSAME(NOFIG,NOTSAM).AND.
       1       UNSAME(NOFIG,LESSN).AND.UNSAME(NOFIG,LESSEQ).AND.
       2       UNSAME(NOFIG,GREATR).AND.UNSAME(NOFIG,GRAE 下)
       1       UNSAME(NOFIG,GRATE).AND.UNSAME(NOFIG,GRATLQ)) GO TO 730
       CALL OUTCOD(NOFIG,OUTPUT,LL)
       IK=IK-1
       GO TO 800

730   IF(UNSAME(NOFIG,LSBSCP)) GO TO 800
       IK=IK-1
       IF(UNSAME(INPUT(JK),RTPREN)) GO TO 8730
       CALL OUTCOD(RTPREN,OUTPUT,LL)
       JK=JK+1
       GO TO 266

C
C 800 IS THE LABEL OF RELATION.
C
C 800   IF(UNSAME(N(IK),NEGATE))GO TO 850
       CALL OUTCOD(NEGATE,OUTPUT,LL)
       IK=IK-1

C     850 IS NEGATION.
C
C 850   IF(UNSAME(N(IK),AND)) GO TO 870
       CALL OUTCOD(SCNAND,OUTPUT,LL)
       IK=IK-1

C     870 IS THE LABEL OF DISJUNCTION.
C
C 870   IF(UNSAME(INPUT(JK),AND)) GO TO 880
       CALL STORE(AND,N,IK,JK)
       CALL OUTCOD(FSTAND,OUTPUT,LL)
       GO TO 100

880   IF(UNSAME(N(IK),OR)) GO TO 900
       CALL OUTCOD(SCNDOR,OUTPUT,LL)
       IK=IK-1

C     900 IS THE LABEL OF CONJUNCTION.
C
C 900   IF(UNSAME(INPUT(JK),OR)) GO TO 910
       CALL STORE(OR,N,IK,JK)
       CALL OUTCOD(FRSTOR,OUTPUT,LL)
GO TO 100
IF(UNSAME(N(IK),TOREAL)) GO TO 1000
CALL OUTCOD(TOREAL,OUTPUT,LL)
IK=IK-1
GO TO 300

1000 IS THE LABEL OF EXPRESSION.

1000
N=IK
IF(UNSAME(NOFIK,ASSINE)) GO TO 1010
CALL OUTCOD(ASSINE,OUTPUT,LL)
IK=IK-1
GO TO 1000

1010 IF(UNSAME(NOFIK,LPFREN)) GO TO 1020
IK=IK-1
IF(UNSAME(INPUT(JK),LTPFREN)) GO TO 9010
JK=JK+1
GO TO 300

1020 IF(UNSAME(NOFIK,TEST)) GO TO 1030
CALL OUTCOD(TEST,OUTPUT,LL)
IK=IK-1
IF(UNSAME(INPUT(JK),THEN)) GO TO 9020
CALL STORE(THEN,N,IK,JK)
GO TO 100

1030 IF(UNSAME(NOFIK,THEN)) GO TO 1040
CALL OUTCOD(SWITCH,OUTPUT,LL)
CALL OUTCOD(THEN,OUTPUT,LL)
CALL OUTCOD(NEGATE,OUTPUT,LL)
CALL OUTCOD(TEST,OUTPUT,LL)
IF(UNSAME(INPUT(JK),ELSE)) GO TO 9030
N(IK)=ELSE
JK=JK+1
GO TO 100

1040 IF(UNSAME(NOFIK,ELSE)) GO TO 1050
CALL OUTCOD(SWITCH,OUTPUT,LL)
CALL OUTCOD(THEN,OUTPUT,LL)
CALL OUTCOD(SEMCLN,OUTPUT,LL)
IK=IK-1
GO TO 1000

1050 IF(UNSAME(NOFIK,LSTHD)) GO TO 1060
IK=IK-1
IF(UNSAME(INPUT(JK),LSTEND)) GO TO 1055
CALL OUTCOD(LSTEND,OUTPUT,LL)
JK=JK+1
GO TO 300

1055 IF(UNSAME(INPUT(JK),COMMA)) GO TO 9055
CALL OUTCOD(COMMA,OUTPUT,LL)
CALL STORE(LSTHD,N,IK,JK)
GO TO 100

1060 IF(UNSAME(NOFIK,PRCEND)) GO TO 1100
IK=IK-1
IF(UNSAME(INPUT(JK),PRCEND)) GO TO 9060
JK=JK+1
CALL OUTCOD(PRCEND,OUTPUT,LL)
GO TO 300

1100 IS THE LABEL OF STATEMENT.

1100 IF(UNSAME(N(IK),SEMCLN)) GO TO 1110
IK=IK-1
1110 ICODE=INPUT(JK)
IF(UNSAME(ICODE,SEMCLN)) GO TO 1120
CALL OUTCOD(SEMCLN,OUTPUT,LL)
CALL STORE(SEMCLN,N,IK,JK)
GO TO 100
1120 IF(UNSAME(ICODE,TRMNAT)) GO TO 9120
1121 CALL OUTCOD(TRMNAT,OUTPUT,LL)
JK=JK+1
IF(UNSAME(N(IK),BEGIN)) GO TO 9121
 IT=STORAJ(BLKNUM)-1
BLKNUM=BLKNUM-1
IK=IK-1
IF(IT.LT.0) GO TO 9121
C
C 1200 IS THE LABEL OF BLOCK.
C
1200 IF(UNSAME(N(IK),KBGIN)) GO TO 1000
IF(UNSAME(INPUT(JK),KEND)) GO TO 9200
I1K=LL+1
DO 1210 I1K=I1K+1000
1210 OUTPUT(I1K)=IBLANK
I1K=0
1220 I1K=I1K+1
I1K=11*I1K
ITT=I1K-10
IF(I1K.GT.1000) I1K=1000
IF(I1K.EQ.1000 OR OUTPUT(I1K+1).EQ.I1BLANK) GO TO 1240
1230 WRITE(1235)(OUTPUT(NOFIK),NOFIK=ITT,I1K)
GO TO 1220
1240 WRITE(1245)(OUTPUT(NOFIK),NOFIK=ITT,I1K)
1235 FORMAT(6X,11A6,8X)
1245 FORMAT(A195X,11A6,8X)
STOP
6000 WRITE(6,6080)
6080 FORMAT(1H0,62HTHE FOLLOWING IS A TRACE OF THE PORTION OF YOUR
1PROGRAM THAT CONTAINED THE ERROR. */56HTHE LAST WORD CAUSED
2THIS ERROR MESSAGE TO BE WRITTEN* )
6020 WRITE(6,6050)(INPUT(MM),MM=MARK,JK)
6050 FORMAT(1H0,12A6)
WRITE(6,6060)
6060 FORMAT(1H0,25HTHE ERROR SCAN CONTINUES.//////////)
6090 JK=JK+1
IF(JK.GT.500) GO TO 9500
ICODE=INPUT(JK)
IF(UNSAME(ICODE,SEMCLN)) GO TO 6095
JK=JK+1
MARK=JK
GO TO 100
6095 IF(UNSAME(ICODE,TRMNAT)) GO TO 6090
MARK=JK
JK=JK+1
GO TO 1110
8000 WRITE(6,4000)
4000 FORMAT(1H0,50HIMPROPER CONTROL SYMBOL AT BEGINNING OF PROGRAM.
1 )
JK=JK+1
GO TO 100
8230 WRITE(6,4230)
4230 FORMAT(1H0,50HILLEGAL OR MALFORMED IDENTIFIER. PROGRAM INVALID
1. )
GO TO 6000
8240 WRITE(6,4240)
4240 FORMAT(1H0,60HILLEGAL PUNCTUATION IN FORMAL DECLARATION. INVAL
1ID PROGRAM. )
GO TO 6000
WRITE(6,4241)
4241 FORMAT(1HO*60HFORMAL DECLARATION IN NONPROCEDURE BLOCK. INVALID PROGRAM. )
GO TO 6000
WRITE(6,4255)
4255 FORMAT(1HO*70HILLEGAL PUNCTUATION IN DECLARATION PART OF BLOCK
1. INVALID PROGRAM. )
GO TO 6000
WRITE(6,4256)
4256 FORMAT(1HO*67HDECLARATION ATTEMPT AT WRONG POINT IN PROGRAM.
1INVALID PROGRAM. )
GO TO 6000
WRITE(6,4261)
4261 FORMAT(1HO*60HATTEMPTED USE OF UNDECLARED IDENTIFIER. INVALID
1PROGRAM. )
GO TO 6000
WRITE(6,4730)
4730 FORMAT(1HO*60HUNBALANCED SUBSCRIPTING PARENTHESSES. INVALID
1PROGRAM. )
GO TO 6000
WRITE(6,5010)
5010 FORMAT(1HO*67HUNBALANCED PARENTHESSES AROUND AN EXPRESSION. INVALID
1PROGRAM. )
GO TO 6000
WRITE(6,5020)
5020 FORMAT(1HO*60HMISSING THEN** IN CONDITIONAL STATEMENT. INVALID
1PROGRAM. )
GO TO 6000
WRITE(6,5030)
5030 FORMAT(1HO*60HMISSING *ELSE* IN CONDITIONAL STATEMENT. INVALID
1PROGRAM. )
GO TO 6000
WRITE(6,5055)
5055 FORMAT(1HO*41HMISSING *COMMA* IN LIST. INVALID PROGRAM. )
GO TO 6000
WRITE(6,5060)
5060 FORMAT(1HO*60HMISSING $. TO END A PROCEDURE DEFINITION. INVALID
1PROGRAM. )
GO TO 6000
WRITE(6,5120)
5120 FORMAT(1HO*670HNO SEMICOLON** OR *END* AT END OF A STATEMENT.
1INVALID PROGRAM. )
GO TO 6000
WRITE(6,5121)
5121 FORMAT(1HO*60HBEGINS* AND *ENDS* DO NOT BALANCE. INVALID
1PROGRAM. )
GO TO 6000
WRITE(6,5200)
5200 FORMAT(1HO*67HIMPROPER CONTROL CARD SYMBOLS. *$* IS MISSING
1AT END OF PROGRAM. )
GO TO 6000
WRITE(6,6050)(OUTPUT(IKK),IKK=1,530)
STOP
END

SUBROUTINE STORE(ITEM,INPLAC,INDEXI,INDEXJ)
DIMENSION INPLAC(300)
INDEXI=INDEXI+1
INPLAC(INDEXI)=ITEM
INDEXJ=INDEXJ+1

$IFT: STOREX

SUBROUTINE OUTCOD(ITEM, INPLAC, INDEXL)
DIMENSION INPLAC(1000)
INDEXL=INDEXL+1
INPLAC(INDEXL)=ITEM
IF(INDEXL+1.GT.1000) GO TO 5
RETURN

5 WRITE(6,10)(INPLAC(INDEXL),INDEXL=1,1000)
10 FORMAT(1HO,5X,12A6)
WRITE(6,11)
11 FORMAT(1HO,5X,31HPROGRAM TOO LARGE TO TRANSLATE.)
STOP

LOGICAL FUNCTION NOTIDT(NAME)
LOGICAL LETTER
IF('NOT'.EQ.(ISHIFT(FSTSYM(NAME)))) GO TO 5
NOTIDT=.FALSE.
RETURN
5 NOTIDT=.TRUE.
RETURN
END

SUBROUTINE EDITOR(INPUT)
C INITIALILATION
LOGICAL DIGIT,LETTER
INTEGER STRGHD
DATA STRGHD/2He/**
INTEGER SEMCLN,PROCMD,PRIOD5,SPACE,BLANK,PERIOD,ASTRSK
DATA ICOMMA,NUMBER/2H0**,6H6,NNBM**/
DATA SEMCLN,PROCMD,LISTHD,PRIODS/2H**,3HC$,2H,,2H0,/**
INTEGER ZERO,ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGHT
DIMENSION INPUT(3000)
DATA ZERO,ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGHT,NINE/
11H0,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9/
DATA IBLANK,ASTRSK/2H0,1H*/**
DATA BLANK,PERIOD,PLUS,MINUS,IASIGN,IFLDIV,IDOAR,
1RPREN,IASTRK/1H*1H,,2H0*,2H0-,2H0!,2H0**,2H0$*,2H0*/
DATA ILPREN,IXPN5/2H0*,2H**/**
DO 10 KK=1,46
LM=66*KK
LL=LM-65
IF(LM.GT.3000) LM=3000
READ(5,11)MARKER,(INPUT(KL),KL=LL,LM)
IF(ISHIFT(MARKER).EQ.IASTRK) GO TO 13
CONT
11 FORMAT(A1,5X,66A1,8X)
12 FORMAT(1H00A1)
13 KK=0
WRITE(6,12)(INPUT(KL),KL=1,LM)
15 KK=KK+1
IF(KK.GT.3000) GO TO 80
ICODE=ISHIFT(INPUT(KK))
IF(ICODE.EQ.IBLANK) GO TO 15
IF(ICODE.EQ.PRIOD5) GO TO 20
IF(ICODE.EQ.IPLUS.OR.ICODE.EQ.IMINUS.OR.ICODE.EQ.IASIGN.OR.ICODE.EQ.ICOMMA.OR.ICODE.EQ.IFLDIV.OR.ICODE.EQ.IDOAR)
15 ICODE=EQ.ICOMMA.OR.ICODE.EQ.IFLDIV.OR.ICODE.EQ.IASIGN.OR.ICODE=EQ.IDOAR) GO TO 25
IF(ICODE.EQ.IDOAR) GO TO 30
IF(ICODE.EQ.IASTRK) GO TO 35
IF(DIGIT(INPUT(KK))) GO TO 45
IF(LETTER(ICODE)) GO TO 40
GO TO 15

C RESERVED WORK BEGINNING WITH PERIOD

20 KL=KL+1
   KK=KK+1
   ICODE=ISHIFT(INPUT(KK))
   IF(ICODE.NE.1) GO TO 21
   INPUT(KL)=SPACE(0, PERIOD)
   LL=1
   INPUT(KL)=IPACK(INPUT(KL), 6, INPUT(KK))
   KK=KK+1
   ICODE=ISHIFT(INPUT(KK))
   IF(ICODE.EQ.IBLANK) GO TO 4
   LL=LL+1
   IF(ICODE.EQ.PRIOD5) GO TO 3
   IF(LL.LT.6) GO TO 24
   KK=KK-1
   GO TO 15
30 INPUT(KL)=IPACK(INPUT(KL), SPACE(6*LL, PERIOD))

C SINGLE CHARACTER OPERATOR

40 KL=KL+1
   INPUT(KL)=INPUT(KK)
   GO TO 15

C FIRST CHARACTER OF OPERATOR WITH PERIOD AS SECOND CHARACTER

30 KL=KL+1
   IF(ISHIFT(INPUT(KK+1)).NE.PRIOD5) GO TO 31
   INPUT(KL)=IPACK(SPACE(0, INPUT(KK)), PRIOD5)
   KK=KK+1
   GO TO 15
31 INPUT(KL)=INPUT(KK)
   GO TO 15

C FIRST ASTERISK OF POSSIBLE ASTERISK PAIR

35 KL=KL+1
   IF(ISHIFT(INPUT(KK+1)).NE.IASTRK) GO TO 36
   KK=KK+1
   INPUT(KL)=IEXPNT
   GO TO 15
36 INPUT(KL)=IASTRK
   GO TO 15

C FIRST LETTER OF AN IDENTIFIER

40 KL=KL+1
   INPUT(KL)=0
   LL=0
41 INPUT(KL)=IPACK(INPUT(KL), SPACE(6*LL, INPUT(KK)))
   KK=KK+1
   LL=LL+1
IF(LLL GT 5) GO TO 43
IF(DIGIT(INPUT(KK)) OR LETTER(ISHIFT(INPUT(KK)))) GO TO 41
DO 42 LLL = LLL + 5
42 INPUT(KL) = IPACK(INPUT(KL), SPACE(6*LLL, BLANK))
43 KK = KK - 1
GO TO 15
C FIRST DIGIT OF NUMBER
45 KL = KL + 1
INPUT(KL) = NUMBER
KL = KL + 1
TEMP = 0.0
46 IF(INPUT(KK) NE ZERO) GO TO 47
ADDEND = 0.0
GO TO 56
47 IF(INPUT(KK) NE ONE) GO TO 48
ADDEND = 1.0
GO TO 56
48 IF(INPUT(KK) NE TWO) GO TO 49
ADDEND = 2.0
GO TO 56
49 IF(INPUT(KK) NE THREE) GO TO 50
ADDEND = 3.0
GO TO 56
50 IF(INPUT(KK) NE FOUR) GO TO 51
ADDEND = 4.0
GO TO 56
51 IF(INPUT(KK) NE FIVE) GO TO 52
ADDEND = 5.0
GO TO 56
52 IF(INPUT(KK) NE SIX) GO TO 53
ADDEND = 6.0
GO TO 56
53 IF(INPUT(KK) NE SEVEN) GO TO 54
ADDEND = 7.0
GO TO 56
54 IF(INPUT(KK) NE EIGHT) GO TO 55
ADDEND = 8.0
GO TO 56
55 IF(INPUT(KK) NE NINE) GO TO 56
ADDEND = 9.0
56 TEMP = ADDEND + 10.0*TEMP
KK = KK + 1
GO TO 46
57 IF(INPUT(KK) EQ PERIOD) GO TO 57
GO TO 68
58 SCALE = 1.0
59 SCALE = SCALE*0.1
KK = KK + 1
IF(INPUT(KK) NE ZERO) GO TO 59
GO TO 58
59 IF(INPUT(KK) NE ONE) GO TO 60
TEMP = SCALE + TEMP
GO TO 58
60 IF(INPUT(KK) NE TWO) GO TO 61
TEMP = 2.0*SCALE + TEMP
GO TO 58
61 IF(INPUT(KK) NE THREE) GO TO 62
TEMP = 3.0*SCALE + TEMP
GO TO 58
62 IF(INPUT(KK) NE FOUR) GO TO 63
TEMP = 4.0*SCALE + TEMP
GO TO 58
63 IF(INPUT(KK).NE.FIVE) GO TO 64
   TEMP=5.0 *SCALE + TEMP
   GO TO 58
64 IF(INPUT(KK).NE.SIX) GO TO 65
   TEMP=6.0 *SCALE + TEMP
   GO TO 58
65 IF(INPUT(KK).NE.SEVEN) GO TO 66
   TEMP=7.0 *SCALE + TEMP
   GO TO 58
66 IF(INPUT(KK).NE.EIGHT) GO TO 67
   TEMP=8.0 *SCALE + TEMP
   GO TO 58
67 IF(INPUT(KK).NE.NINE) GO TO 68
   TEMP=9.0 *SCALE + TEMP
   GO TO 58
68 KK=KK-1
   CALL JUSTOT(TEMP,INPUT(KL),INPUT(KL+1))
   KL=KL+1
   GO TO 15
200 INPUT(KL)=STRGHD
   LL=0
201 KL=KL+1
   INPUT(KL)=0
202 KK=KK+1
   IF(ISHIFT(INPUT(KK)).EQ.IASTRK) GO TO 204
   INPUT(KL)=IPACK(INPUT(KL),SPACE(6*LL,INPUT(KK)))
   IF(LL.GT.5) GO TO 203
   LL=LL+1
   GO TO 202
203 KK=KK-1
   GO TO 15
204 KK=KK+1
   GO TO 4
80 KL=KL+1
DO 81 KK=KL,3000
81 INPUT(KK)=IBLANK
RETURN
END

SIBFTC DIGITX
LOGICAL FUNCTION DIGIT(JIG)
   INTEGER ZERO,ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGHT
   DATA ZERO,ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGHT/NINE/
   1 1H0,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9/
   IF(JIG.EQ.ZERO.OR.JIG.EQ.ONE.OR.JIG.EQ.TWO.OR.JIG.EQ.THREE.OR.
    JIG.EQ.FOUR.OR.JIG.EQ.FIVE.OR.JIG.EQ.SIX.OR.JIG.EQ.SEVEN.OR.
    JIG.EQ.EIGHT.OR.JIG.EQ.NINE) GO TO 8
   DIGIT = .FALSE.
   RETURN
8   DIGIT = .TRUE.
   RETURN
END

$IBFTC LETTERX
LOGICAL FUNCTION LETTER(LET)
   DATA A,B,C,D,E,F,G,H,I,J,K/
   12H0A,2HUB,2HUC,2HOD,2HOE,2HOF,2HOG,2HOI,2HOJ,2HOK/
   12HOL,2HOM,2HON,2HOP,2HOQ,2HOR,2HOS,2HOT,2HOU,2HOU,2HOV,2HOW/
   22H0X,2H0Y,2H0Z/
   IF(LET.EQ.A.OR.LET.EQ.B.OR.LET.EQ.C.OR.LET.EQ.D.OR.LET.EQ.E.OR.
1 LET.EQ.F OR LET.EQ.G OR LET.EQ.H OR LET.EQ.I OR LET.EQ.J OR LET.EQ.K OR LET.EQ.L OR LET.EQ.M OR LET.EQ.N OR LET.EQ.O OR LET.EQ.P OR LET.EQ.Q OR LET.EQ.R OR LET.EQ.S OR LET.EQ.T OR LET.EQ.U OR LET.EQ.V OR LET.EQ.W OR LET.EQ.X OR LET.EQ.Y OR LET.EQ.Z
5 GO TO 8
    LETTER = .FALSE.
    RETURN
8 LETTER = .TRUE.
    RETURN
END

$IBMAP SPACEX 7
ENTRY SPACE
* SPACE(BitStoRight,WordToBeShifted)
SPACE CLA* 3,4 FETCH SHIFTING NUMBER
    STA ++3
    CAL* 4,4 STORES 5,1-35 IN AC, Bits P, 1-35
    ANA MASK AND WORDS 5,1-35 WITH AC, P, 1-35
    ARS ** SHIFT RIGHT THE AC
    SLW ;M
    CLA MM
    TRA 1,4
MM BSS 1
    MASK OCT 7700000000000 2-7,5,10-0's
END

$IBMAP IPK
ENTRY IPACK
* TAKES THE LOGICAL OR OF TWO ALPHANUMERIC WORDS
IPACK CAL* 3,4
    ORA* 4,4
    SLW MM
    CLA MM
    TRA 1,4
MM BSS 1
END

$IBMAP JUST
ENTRY JUSTOT
* JUSTOT(FL, A, B) = THE ABSOLUTE VALUE OF FL
* IS TRANSFORMED TO 2 BCD WORDS AND
* STORED IN A AND B
JUSTOT SXA BACK, 1
    CLA* 3,4
    SSP
    XCA
    PXA 0,0
    AXT 6,1
    ALS 3
    LGL 3
    TIX *-2,1,1
    STO* 4,4
    PXA 0,0
    AXT 6,1
    ALS 3
    ALS 3
    TIX *-2,1,1
    STO* 5,4
BACK AXT **,1
    TRA 1,4
END

$IBMAP ISHFT
* SHIFTS THE ARGUMENT RIGHT ONE CHARACTER AND RETURNS VALUE
ENTRY ISHIFT
ISHIFT CAL* 3,4
ARS 6
TRA 1,4
END
$IBMAP UNSAMEX
ENTRY UNSAME
JNSAM: CAL* 3,4
ERA* 4,4
TZE 1,4
CLA =1
TRA 1,4
END
$IBMAP FSTSYM
ENTRY FSTSYM
FSTSYM CAL* 3,4
ANA MASK1
ORA MASK2
SLW TEMP
CLA TEMP
TRA 1,4
MASK1 OCT 770000000000
MASK2 OCT 006060606060
TEMP BSS 1
END
$ENTRY TRANSL
$DATA
ENTRY *BEGIN*N*FIBO*N*CUT*_* N=N=10)*)
FIBO= *$FRML*K*IF*K*LEQ* 2 THEN*1 ELSE*
FIBO*(K-1)+FIBO*(K-2)$$ OUT*(*FIBO= FIBO)$$
OUT*(*FIBO(*N***) = FIBO*(N)*)
* END*EXIT*
Appendix 4. The EULER Polish Interpreter Listing
INTEGER T, FORMAL, VARIABL, BLKNUM, STORAJ, ASSINE,
BEGIN, TRMNT, SEMCLN, SYMBOL, COMMA, RSHSCP, CONCAT, TAIL, CUT,
2TEST, SWITCH, THEN, FRSTCR, SCNDOR, STAND, SCAND, ADD,
3SUBTRC, FLTDIV, EXPNNT, PLUS, TOREAL, TOINTG, TOABS1, PRCHI,
4PREDNT, TRUTH, FALSHTY, GREATER, GRATEQ, UNDFND, PRCHD
DIMENSION LAVS(2000), LSTRTN(2000), IONLTS(1500), CPM(0)(500),
1 IPKND(500), STORAJ(200), JUMPBK(200), PRAKLS(200)
COMMON/AVSLX, AVSL, CODE(500)
INTEGER CODE
DATA ICOMMA, LEPREN, IRPREN/1H, 6H, (1H)/
DATA LUCAL, ISWM, TSLGCL, ISLAPL, ISLIST, ISLUNDF, ISRF*/
16H0, EGCL, 6H0, ISNU, 6H0, ISLC, 6H0, ISLA, 6H0, ISLI, 6H7, ISL, 6H0, ISRF/
DATA TRMNT, BEGIN, SEMCLN, NEW, LABEL, LABLEF, ASSINE, JUMP/
16H0, ENDC, 6H0, BEGIN, 3H0, , 6H0, NEW, 6H0, LABL, 6H0, LBREF, 2HO, = 6H0, G0TC/
DATA CUT, TEST, SWITCH, THEN, NEGATE, FRSTCR, SCNDOR/
16H0, OLT, 5HO, IF, 6H0, SWCH, 6H0, THEN, 6H0, NOT, 6H0, FSOR,
26H0, SCOR/
DATA LESSEG, ADD, SUBTRC, PLUS, NEG, MODULO, MLTPLY, INTCIV/
15H0, LEQ, 2H0, 2H0, 6H0, PLUS, 6H0, NEG, 6H0, MDLO, 2H0, * 4H0, /
DATA FLTDIV, EXPNNT, CONCAT, INSIDE, UNDFND, TAIL, LSCALL, LIST/
12H0, 3H0, 6H0, CNCT, 5H0, IN, 6H0, UNDF, 6H0, TAIL, 6H0, LSLC, 6H0, LIST/
DATA ISREC, SYMBl, SYMBOL, RSHSCP, VARIABL/
16H0, ISPR, 6H0, ISSL, 3H0, *, 2H0, 1H0, 6H0, VRBL/
DATA FSTAND, SCNANL, ISSAME, NOTSAM, GREATR, LESSTN, GRATEQ/
16H0, FAND, 6H0, SAND, 5H0, Eq, 6H0, NEQ, 5H0, GT, 5H0, LT, 6H0, GEQ/
DATA PRCCHD, PRCND, FORMAL, LABEL, LISTHD, COMMA, LISTEND/
13H0, *, 3H0, 6H0, FRML, 6H0, LABL, 3H0, (2H0, 3H0)/
DATA NUMBER, TOREAL, LENGTH, TOINTG, TOABS1/
16H0, NMBR, 6H0, REAL, 6H0, LNTG, 6H0, INTG, 6H0, ABST/
DATA TRUTH, FALSHTY, 6H0, TRUF, 6H0, FLSE/
CALL INITIAS(LAVS, 2000)
DO 1 IK=1, 46
    T=11*IK
    I=T-10
1 IF(I.GT.500) T=500
READ(5, 2) ITEMP, (CGDE(INDEX), INDEX=1, T)
    FORMAT(A1, 5X, 11A6, 8X)
    IF(ISHIFT(ITEMP).EQ. MLTPLY) GO TO 5
2 CONTINUE
5 WRITE(6, 9006) (CODE(I), I=1, T)
WRITE(6, 9005)

I=0
    IK=0
    IBASH=0
    BLKNUM=0
    JMPRTN=0
I=0
10 I=I+1
CALL CUTOUT(7, CODE(I), IBASH)
ICODE=ISHIFT(CODE(I))
    IF(ICODE .EQ. LSCALL) GO TO 300
    IF(ICODE .EQ. SYMBOL) GO TO 305
    IF(ICODE .EQ. TEST) GO TO 430
    IF(ICODE .EQ. SWITCH) GO TO 440
    IF( ICODE .EQ. NEGATE ) GO TO 445
```
IF(CODE.EQ.0) THEN) GO TO 10
IF(CODE.EQ. LIST) GO TO 500
IF(CODE.EQ. CONCATE) GO TO 505
IF(CODE.EQ. TAIL) GO TO 510
IF(CODE.EQ. LENGTH) GO TO 515
IF(CODE.EQ. LISTS HD) GO TO 600
IF(CODE.EQ. COMMA) GO TO 605
IF(CODE.EQ. LSTEND) GO TO 615
IF(CODE.EQ. RSBSCP) GO TO 650
IF(CODE.EQ. BEGIN) GO TO 700
IF(CODE.EQ. TRMNAT) GO TO 705
IF(CODE.EQ. SEMCLN) GO TO 710
IF(CODE.EQ. VARI BL) GO TO 715
IF(CODE.EQ. NUMBER) GO TO 725
IF(CODE.EQ. UNDF ND) GO TO 727
IF(CODE.EQ. NEW) GO TO 730
IF(CODE.EQ. LABDEF) GO TO 731
IF(CODE.EQ. LABEL) GO TO 735
IF(CODE.EQ. ASSINE) GO TO 740
IF(CODE.EQ. OUT) GO TO 755
IF(CODE.EQ. FRSTOR) GO TO 775
IF(CODE.EQ. SCNAND) GO TO 780
IF(CODE.EQ. SCAND) GO TO 780
IF(CODE.EQ. FNSTD) GO TO 785
IF(CODE.EQ. PLUS) GO TO 10
IF(CODE.EQ. NEG) GO TO 781
IF(CODE.EQ. ISSAME) GO TO 790
IF(CODE.EQ. NOTSAM) GO TO 795
IF(CODE.EQ. GREATER) GO TO 796
IF(CODE.EQ. LESSER) GO TO 797
IF(CODE.EQ. GRATEQ) GO TO 798
IF(CODE.EQ. LESSEQ) GO TO 799
IF(CODE.EQ. ADD) GO TO 805
IF(CODE.EQ. SUBTRC) GO TO 807
IF(CODE.EQ. MULPLY) GO TO 810
IF(CODE.EQ. FLT DIV) GO TO 811
IF(CODE.EQ. INT DIV) GO TO 812
IF(CODE.EQ. MODULO) GO TO 813
IF(CODE.EQ. EXPNNT) GO TO 816
IF(CODE.EQ. TRUTH) GO TO 817
IF(CODE.EQ. FALSTY) GO TO 819
IF(CODE.EQ. LOGCAL) GO TO 820
IF(CODE.EQ. PROC HD) GO TO 840
IF(CODE.EQ. INSIDE) GO TO 845
IF(CODE.EQ. PRCEND) GO TO 865
IF(CODE.EQ. ISMNR) GO TO 860
IF(CODE.EQ. IS LGCL) GO TO 861
IF(CODE.EQ. ISL ABL) GO TO 862
IF(CODE.EQ. IS LIST) GO TO 863
IF(CODE.EQ. ISU NDF) GO TO 864
IF(CODE.EQ. ISPRCD) GO TO 866
IF(CODE.EQ. ISYM BL) GO TO 867
IF(CODE.EQ. ISRFRN) GO TO 868
IF(CODE.EQ. FORMAL) GO TO 870
IF(CODE.EQ. JUMP) GO TO 930
IF(CODE.EQ. TOREAL) GO TO 937
```
IF(ICODE .EQ. TOINTG) GO TO 938
IF(ICODE .EQ. TOABSL) GO TO 939
WRITE(6,9931)
STCP

C Assumes varibl has placed a link to the datum in CPRND(IK)
300 LINK = LNKR(CPRND(IK))
IF,ID(INHALT(LINK)).NE.6) GO TO 886C
IOPRNC(IK) = 6
CALL STRDIR(CCNY(LINK+1),OPRND(IK))
GO TO 10

305 I=I+1
IK=IK+1
IOPRNC(IK)=7
CALL STRDIR(CODE(I),OPRND(IK))
GO TO 10

C Test CPERAND for truth.  If false, scan ahead until a matching
C then is found.  If true, continue evaluation.
430 IF(CPRND(IK).EQ.1.0) GO TO 10
ITT=0
431 I=I+1
ICODE=ISHIFT(CODE(I))
IF(ICODE.EQ.TEST)ITT=ITT+1
IF(ICODE.EQ.THFN)GO TO 432
GO TO 431

432 IF(ITT.EQ.0) GO TO 10
ITT=ITT-1
GO TO 431

C Exchange topmost Cperands during a conditional statement.
440 TEMP=OPRND(IK-1)
ITT=IOPRND(IK-1)
OPRND(IK-1)=CPRND(IK)
IOPRND(IK-1)=IOPRND(IK)
CPRND(IK)=TEMP
IOPRND(IK)=ITT
GO TO 10

C Negate topmost operand for conditional statement and logical
C negation.
445 IF(IOPRND(IK).NE.1) GO TO 8775
OPRND(IK)=1.0-OPRND(IK)
GO TO 10

C Creation of a list of length OPRND(IK).
500 IF(IOPRND(IK).NE.0) GO TO 8806
INDEX=OPRND(IK)
ICPRND(IK)=3
IF(INDEX.NE.0) GO TO 501
CALL STRDIR(I,OPRND(IK))
GO TO 10

501 IVRBL=NUCELL(DUMMY)
IF(INDEX.EQ.1) GO TO 503
LLCELL=IVRBL
DC 502 ITEMP=2, INDEX
LRCELL=NUCELL(DUMMY)
CALL SETIND(4,-1,LRCELL,LLCELL)
C THE NUCELL ROUTINE ZERORES THE LNRK FIELDS.
502 LLCELL=LRCELL
503 CALL SETDIR(0,0,IVRBL,OPRND(IK))
GO TO 10.

C CONCATENATION OF TWO LISTS. A LINK TO THE RESULTING LIST IS
C RETURNED TO THE TCP OF THE OPRND STACK. THE TWO CONCATENATED LISTS
C ARE COPIED BY THE COPY ROUTINE. THE COPY ROUTINE HAS PROVISIONS
C FOR LISTS THAT HAVE ZERO REFERENCE COUNT (I.E., NO NAME) AND FOR
C EMPTY LISTS.

505 IF(IOPRND(IK).NE.3.OR.IOPRND(IK-1).NE.3) GO TO 8510
LINKO=LNRK(OPRND(IK-1))
CALL COPY(LINKO, ITOP, IBOT)
LINKO=LNRK(OPRND(IK))
CALL COPY(LINKO, JTOP, JBOT)
IK=IK-1
CALL SETDIR(ITOP, OPRND(IK))
CALL SETIND(-1,-1, JTOP, IBOT)
GO TO 10

C TAKES THE SUFFIX OF A LIST, IGNORES REFERENCE COUNT OF
C FIRST CELL, AND PUTS LINK TO THAT CELL ON TOP OF OPERAND LIST

510 ITEMP=LNRK(OPRND(IK))
IF(IOPRND(IK).NE.3) GO TO 8510
IF(ITEMP.LE.0) GO TO 8511
ITEMP=LNRK(INHALT(ITEMP))
IF(ITEMP.LE.0) GO TO 8650
CALL SETDIR(ITEMP, OPRND(IK))
CALL SETIND(-1, LNKI(INHALT(ITEMP))+1,-1,ITEMP)
GO TO 10

C FINDS THE LENGTH OF THE LIST WHOSE POINTER IS AN OPERAND.

515 IF(IOPRND(IK).NE.3) GO TO 8510
TEMP=0.0
ITEMP=LNRK(OPRND(IK))
IOPRND(IK)=0
516 IF(ITEMP.LE.0) GO TO 517
TEMP=TEMP+1.0
ITEMP=LNRK(INHALT(ITEMP))
GO TO 516
517 OPRND(IK)=TEMP
GO TO 10

C A PLACE IS RESERVED ON THE OPERAND LIST FOR A LINK.

600 IK=IK + 1
IOPRND(IK) = 3
CALL STRDIR(LISTHC, OPRND(IK))
GO TO 10

C TYPED INFORMATION ON TOP OF THE OPERAND LIST IS REPLACED BY
C AN INTEGER LINK TO A LIST CELL THAT CONTAINS
C THIS INFORMATION. THE IK COUNT IS NOT DECREMENTED AND THE
C SUBLIST REFERENCE COUNTERS ARE INCREMENTED.
C
605      IKK=IOPRND(IK)
          JJ=LNKR(OPRND(IK))
       IF(IKK.NE.3.OR.JJ.LE.0) GO TO 610
          CALL SETIND(-1, LNKI(INHALT(JJ)))*1J-1, JJ)

610      ITEMP=NUCELL(DUMMY)
          CALL SETIND(IKK, O, 0, ITEMP)
          CALL STRIND(OPRND(IK), ITEMP+1)
          CALL STRDIR(ITEMP, OPRND(IK))
          IOPRND(IK)=3
          GO TO 10

C PERFORMS THE COMMA OPERATION ON TOPMOST DATUM IN OPERAND STACK.
C THEN PROCEEDS DOWN THE OPERAND STACK TO LINK TOGETHER OPERANDS
C UNTIL LISTHEAD **.** IS ENCOUNTERED AS AN OPERAND. FINALLY, CHECKS
C TO SEE IF THE RESULTING LIST IS THE PARAMETER LIST OF A PROCEDURE
C CALL.
C
615      IF(INTRSC(OPRND(IK), LISTHC)*NE.0) GO TO 620
          CALL STRDIR(0, OPRND(IK))
          GO TO 10

620      ITEMP=NUCELL(DUMMY)
          IKK=IOPRND(IK)
          JJ=LNKR(OPRND(IK))
       IF(IKK.NE.3.OR.JJ.LE.0) GO TO 630
          CALL SETIND(-1, LNKI(INHALT(JJ)))*1J-1, JJ)

630      CALL SETIND(IKK, O, 0, ITEMP)
          CALL STRIND(OPRND(IK), ITEMP+1)
          CALL STRDIR(ITEMP, OPRND(IK))
          IOPRND(IK)=3
          GO TO 645

645      IF(INTRSC(OPRND(IK-1), LISTHC)*EQ.0) GO TO 646
          CALL SETIND(-1, O, LNKI(OPRND(IK)), LNKI(OPRND(IK-1))))
          IK=IK-1
          GC TO 645

646      IK=IK-1
          OPRND(IK)=OPRND(IK+1)
       IF(IOPRND(IK-1)*NE.0) GO TO 10

C THE FORMAL ROUTINE EXPECTS TO FIND A LINK TO THE FIRST
C CELL OF AN ID-DATUM PAIR ON PRAMLs. FORMAL CHECKS FOR
C LISTED AND PARAMETER MATCHING. THE PARAMETER LIST IS LOCAL
C TO THE CALLING BLOCK OF THE PROCEDURE AND IS DEMOLISHED BY END.

JMPRTN=JMPRTN+1
JUMPPEK(JMPRTN)=I
T=I+1
ITT=3*T
CALL STRDIR(OPRND(IK), PRAMLs(JMPRTN))
CALL STRDIR(OPRND(IK), IONLs(ITT))
CALL SETDIR(3,1,0,INH(ITT-1))
I=OPRND(IK-1)
IK=IK-2
BLKNUM=BLKNUM+1
STORAJ(BLKNUM)=T+1
GO TO 10

C THE RIGHT BRACKET ACTS AS AN OPERATOR SO THAT THE TOPMOST OPERAND
C SPECIFIES THE CELL OF A LIST WHOSE LINK IS THE NEXT OPERAND
C DOWN. A REFERENCE (LINK) IS RETURNED TO THE NEXT OPERAND DOWN,
C PRESERVING THE BLOCKNUMBER FOUND BY VARBL.

C
650 INDEX=OPRND(IK)
   IK=IK-1
   IF(INDEX.LE.0) GO TO 10
   LINK=LNRK(INH(1+LNRK(OPRND(IK))))
651 IF(LINK.LE.2) GO TO 650
   INDEX=INDEX-1
   IF(INDEX.EQ.0) GO TO 652
   LINK=LNRK(INH(LINK))
   GO TO 651
652 CALL SETDIR(-1,-1,LINK,OPRND(IK))
   GO TO 10
C
700 BLKNUM = BLKNUM + 1
   STORAJ(BLKNUM) = T+1
   GO TO 10
C THIS IS THE BLOCK END ROUTINE. ITS MAIN FEATURE IS A SEARCH
C FOR LISTS TO RETURN TO LAVS. THE BLOCKNUMBER IS DECREMENTED,
C AND, IF EQUAL TO ZERO, THE PROGRAM TERMINATES.
C
705 INDEX = STORAJ(BLKNUM)
706 IDNTLS(IIT)=0
708 IIT=3*I
   IDNTLS(IIT-1)=0
   IDNTLS(IIT-2)=0
   T = T-1
   IF(T.GE.INDEX) GO TO 706
709 BLKNUM = BLKNUM - 1
   IF(BLKNUM *GT. 0) GO TO 10
   STOP
C SEMICLCN REMOVES THE TOPMOST OPERAND OF THE OPERAND LIST.
C IF THIS OPERAND IS A LINK TO SOME LIST WHOSE FIRST CELL HAS
C ZERO REFERENCE COUNT ERASE IS CALLED.
C
710 IF(OPRND(IK) .NE. 3) GO TO 713
   LLNK = LNRK(OPRND(IK))
   IF(LLNK.LE.0) GO TO 713
   LCOUNT=LNRK(INH(LINK))
   IF(LCOUNT *GT. 0) GO TO 713
   CALL ERASE(LLNK)
IT IS ASSUMED THAT LABELS AND REFERENCES ARE STORED IN THE LNKR
FIELDS OF WORDS AND THAT THEIR DYNAMIC BLOCKNUMBER IS STORED IN
THE LNKL FIELDS. THE NAME FOLLOWING THE VARSB. COMMAND IS LOOKED
UP ON THE IDENT STACK, AND A LINK TO IDENT(3*I'-1), FOR SOME I', IS
PLACED ON TOP OF THE CPRND STACK.

I = I + 1
ITEMP = T
IVREL = CODE(I)

IF (INSTR(IVREL, IDNTLS(3*ITEMP-2)).EQ.0) GO TO 717
ITEMP = ITEMP - 1
IF (ITEMP.LT. 0) GO TO 8716
GO TO 716

LLNK = MAOV(3*ITEMP-1)
CALL SETDIR(0, LNKR(INHALT(LLNK)), LNK, CPRND(IK))
IOPRND(IK) = 5
GO TO 10

T = T + 1
IK = IK + 1
CALL JUSTIN(IOPRND(IK), CODE(I-1), CODE(I))
ICPRND(IK) = 0
GO TO 10

I = I + 1
T = T + 1
ITT = 3*T
IDNTLS(I TT-2) = CODE(I)
CALL SETDIR(4, 0, BLKNUM, IDNTLS(I TT-1))
IDNTLS(I TT) = 0
GO TO 10

I = I + 1
GO TO 10

THIS ROUTINE SEARCHES FOR THE LOCATION OF THE
LABEL, THEN PLACES THE LINK TO THIS LABEL, EXPRESSED AS A
VALUE OF I, INTO IDNTLS(T, 3) ALONG WITH ITS BLOCKNUMBER.

T = T + 1
ITT = 3*T
I = I + 1
KLRL = CODE(I)
IDNTLS(I TT-2) = KLRL
CALL SETDIR(2, O, BLKNUM, IDNTLS(I TT-1))
INDEX = I
ITEMP = 0

INDEX = INDEX + 1
ICODE = ISHIFT(CODE(INDEX))
SCHNEIDER, VI
EULER - EFN SOURCE STATEMENT - IFN(S) -

IF (ICODE EQ BEGIN) ITEMP = ITEMP + 1
IF (ICODE EQ TRMNT) ITEMP = ITEMP - 1
IF (ITEMP LT 0) GO TO 8736
IF (ICODE NE LABCDF) GO TO 736
INDEX = INDEX + 1

IF (ICODE (INDEX) NE KLBL OR, ITEMP NE 0) GO TO 736
INDEX = INDEX + 1
737 IF (ISHIFT (CODE (INDEX )) NE LABCDF) GO TO 738
INDEX = INDEX + 2
GO TO 737.

CALL SETDIR (0, BLKNUM, INDEX, IDNTLS(ITT))
GO TO 10
C

THE ASSIGNEE IS A REFERENCE (TYPE 5) BECAUSE OF THE UNIFORM C
CONVENTIONS OF TREATING THE SEQUENCE **VRBL** NAME** AND THE C
RESULT OF USING RSBSHC ON A LIST FOR SUBSCRIPTING BOTH AS C
REFERENCE POINTERS. THE LINKLEFT FIELD OF THE ASSIGNEE OPERAND C
IS ITS DYNAMIC BLOCKNUMBER, AND THE LINKRIGHT FIELD POINTS TO THE C
IDNTLS OR LIST CELL THAT PRECEDES ITS DATUM IN COMPUTER MEMORY.
C
IF THE ASSIGNED OPERAND IS A REFERENCE OR A LABEL, ITS LINKLEFT C
FIELD ALSO CONTAINS THE DYNAMIC BLOCKNUMBER. IN THIS CASE, THE C
ASSIGNMENT STATEMENT WILL NOT BE EXECUTED IF THE ASSIGNEE IS IN A C
BLOCK GLOBAL TO THE BLOCK OF THE ASSIGNED.

740 IF (IOPRND (IK-1) NE 5) GO TO 8740
INDEX = IOPRNC (IK)
LRNK = LNKRI (IOPRND (IK))
LINK = LNKRI (IOPRND (IK-1))
IF (INDEX EQ 5 OR INDEX EQ, 2) GO TO 745

741 IF (INDEX NE 3 OR LRNK LE 0) GO TO 741
CALL SETIND (-1, LNKLI (INHALT (LRNK)) + 1, -1, LRNK)
C LOCAL AND GLOBAL LISTS CAN BE ASSIGNED TO ONE ANOTHER
CALL SETDIR (0, 0, -1, OPRND (IK))

742 CALL SETIND (INDEX, -1, -1, LINK)
CALL STRIND (OPLRND (IK), LINK + 1)
OPLRND (IK-1) = OPRND (IK)
IOPRND (IK-1) = IOPRND (IK)
IK = IK - 1
GO TO 10

745 IF (LNKLCEOPLRND (IK)) GT. LNKL (OPLRND (IK-1)) GO TO 8745
GO TO 741
C

THIS IS THE WRITEOUT ROUTINE. THE TOP DATUM ON THE OPRND STACK IS C
WRITTEN OUT, AND THIS DATUM IS LEFT UNTouched ON THE OPRND STACK.

755 WRITE (6, 774)
ITT = 0
ITEMP = IOPRND (IK)
IF (ITEMP NE 3) GO TO 761
C 761 IS THE LABEL CORRESPONDING TO A NON-LIST OUTPUT.

AN EMPTY LIST HAVING ZERO LINK IN O(IK) IS TREATED SEPARATELY.

WRITE (6, 770)
WRITE(6,774) 
IBASH=0 
GO TO 10

756 LSTCNT=0
LLNK=LNRK(OPRND(IK))
C LEFT PARENTHESES OF OUTER LIST IS WRITTEN.
CALL CUTOUT(7,LFPREN,ITT)
757 INLNK=INHALT(LLNK)
IF(ID(INLNK),NE.3)GO TO 758
C LSTCNT GIVES DEPTH INTO LIST STRUCTURE WRITTEN OUT.
LSTCNT=LSTCNT+1
C DESCEMT INTO A SUBLIST
LSTRTN(LSTCNT)=LLNK
LLNK=LNRK(INHALT(LLNK+1))
CALL CUTOUT(7,LFPREN,ITT)
GO TO 757
C PRINTOUT OF A DATUM. CHARACTERS NOT SEPARATED BY COMMAS HERE.
758 CALL CUTOUT(ID(INLNK),INHALT(LLNK+1),ITT)
759 IF(LNRK(INLNK),GT.0)CALL OUTOUT(7,ICOMMA,ITT)
760 CALL OUTOUT(7,IPREN,ITT)
761 IF(LSTCNT.LE.0)GO TO 763
C RISE FROM SUBLIST TO MASTER LIST.
LLNK=LNRK(INHALT(LSTRTN(LSTCNT)))
LSTCNT=LSTCNT-1
IF(LLNK.EQ.0)GO TO 760
CALL OUTOUT(7,ICOMMA,ITT)
GO TO 757
C WRITEOUT OF NEXT ELEMENT OF A LIST.
762 LLNK=LNRK(INLNK)
GO TO 757
C END OF LIST WRITEOUT.
763 WRITE(6,774)
IBASH=0
GO TO 10
C WRITEOUT OF NORMAL NONLIST ELEMENT.
761 ICCNT=2
CALL OUTOUT(ITEMP,OPRND(IK),ICONT)
WRITE(6,774)
IBASH=0
GO TO 10
770 FORMAT(1HO,1X,3H( ))
774 FORMAT(1HO/
C C THIS ROUTINE CHECKS THE TOPMOST OPERAND FOR TRUTH OR FALSETY.
C IF TRUE, SKIPS TO SCNDOR+1. IF FALSE, DELETES OPERAND
C AND PROCEEDS TO EVALUATE SECOND OPERAND. SECONDO
C FINDS AN OPERAND THAT IT LEAVES UNTOUCHED.
C
775 IF(IOPRND(IK).NE.1) GO TO 775
776 IF(CPRND(IK),EQ.1.0) GO TO 777
IK=IK-1
GO TO 10
777 ITT=0
778 I=I+1
ICODE=IShift(CODE(I))
IF(ICODE.EQ.1) FRSTOR ITT=ITT+1
IF(ICODE.EQ.2) SCNDOR GO TO 779
GO TO 778
779 IF(ITT.EQ.0) GO TO 10
ITT=ITT-1
GO TO 778

C 780 IF(IOPRND(IK).NE.1) GO TO 8775
GO TO 10
C THIS ROUTINE CHECKS TOPMOST OPERAND FOR TRUTH OR FALSETY.
C IF FALSE, SKIPS TO SCNAND+1. IF TRUE, DELETES OPERAND
C AND PROCEEDS TO EVALUATE SECOND OPERAND.
C 785 IF(IOPRND(IK).NE.1) GO TO 8775
786 IF(CPND(IK).EQ.0.0) GO TO 787
IK=IK-1
GO TO 10
787 ITT=0
788 I=I+1
ICODE=IShift(CODE(I))
IF(ICODE.EQ.1) FSTAND ITT=ITT+1
IF(ICODE.EQ.2) SCNAND GO TO 789
GO TO 788
789 IF(ITT.EQ.0) GO TO 10
ITT=ITT-1
GO TO 788

C A LINK IS PLACED TO PROCHE. SKIP TO PRCEND+1.
C
840 IK=IK+1
OFRND(IK)=I
IOPRND(IK)=6
INDEX=0
841 I=I+1
ICODE=IShift(CODE(I))
IF(ICODE.EQ.1) PROCHD INDEX=INDEX+1
IF(ICODE.EQ.2) PRCEND GO TO 842
GO TO 841
842 IF(INDEX.LE.0) GO TO 10
INDEX=INDEX-1
GO TO 841

C PROVIDES A RETURN JUMP AND DEMOLITION OF BLOCK STORAGE VIA
C THE ORDINARY END ROUTINE.
C ALSO DESTROYS PARAMETER LIST OF THE PROCEDURE IF IT HAS ONE.
C
865 I=JUMPBK(JMPRTN)
JMPRTN=JMPRTN-1
GO TO 705
C 705 IS WHERE THE END ROUTINE IS.
C
C INITIATES PROCEDURE CALLS HAVING NO PARAMETERS AND FETCHES DATUM
C OF NON-PROCEDURE NAMES INTO OPRND(IK).
C
845  LINK = LNKR(OPRND(IK))
     ICIC = ID(INHALT(LINK))
     IOPRN(IK) = IDID
     IF(IDID .NE. 6) GC TO 850
     IK = IK - 1
     JMPRTN = 1 + JMPRTN
     JUMPRT(JMPTNN) = I
     PRAMLS(JMPRTN)=0.0
     BLKNUP = 1 + BLKNUM
     STORA(1,BLKNUM) = 1 + T
     I = CNT(LINK + 1)
     IF(ISHIFT(CODE(I+1)).EQ.FORMAL)GO TO 8845
     GO TO 10
850  CALL STRDIR(INHALT(LINK+1),OPRND(IK))
     GO TO 10

C FORMAL CALLS FOR THE LINK OF A LIST OF PARAMETERS STORED ON TOP
C OF PRAMLS. IF THIS LIST OF PARAMETERS IS TOO SHORT THE UNMATCHED
C PROCEDURE PARAMETERS ARE UNDEFINED. THIS PARAMETER LIST HAS BEEN
C MADE LOCAL TO THE CALLING BLOCK BY LISTEND. NO REFERENCE COUNTERS
C NEED TO BE INCREMENTED FOR ANY POSSIBLE LIST PARAMETERS, SINCE THE
C LIST CONSTRUCTION HAS ALREADY DONE THIS.

870  LINK=LNKR(PRAMLS(JMPRTN))
     I=I+1
     T=T+1
     CALL STRDIR(CODE(I),IDNTLS(3*T-2))
     IF(LINK .EQ. C) GO TO 875
     CALL STRDIR(INHALT(LINK),INHLNK)
     CALL STRDIR(INHLNK,IDNTLS(3*T-1))
     IF(ID(INHLNK).NE.2.AND.ID(INHLNK).NE.5)GO TO 872
     CALL SETDIR(-1,-1,LNKL(INHALT(LINK+1)),IDNTLS(3*T-1))

C THE BLOCK NUMBER OF THE REFERENCE OR LABEL IS BROUGHT TO
C IDNTLS(T, 2)
872  CALL STRDIR(INHALT(LINK+1),IDNTLS(3*T))
     CALL STRDIR(INHLNK,PRAMLS(JMPRTN))
     GO TO 10
875  IDNTLS(3*T)=0
     CALL SETDIR(4,0,0,IDNTLS(3*T-1))
     GC TO 10

C 805  OPRND(IK-1)=OPRND(IK-1)+OPRND(IK)
806  IF((OPRND(IK-1)).NE.0.OR.(OPRND(IK)).NE.0) GC TO 8806
     IK=IK-1
     GO TO 10
807  OPRND(IK-1)=OPRND(IK-1)-OPRND(IK)
     GO TO 806
810  OPRND(IK-1)=OPRND(IK-1)*OPRND(IK)
     GO TO 806
811  IF(OPRND(IK).EQ.0.0) GC TO 8811
     OPRND(IK-1)=OPRND(IK-1)/OPRND(IK)
     GO TO 806
812  IF(OPRND(IK).EQ.0.0) GC TO 8811
     ITEMP=OPRND(IK-1)/OPRND(IK)
     OPRND(IK-1)=ITEMP
     GC TO 806
R2 = CPRND(IK)
IF(R2.EQ.0.0) GO TO 8811
C     OPRND(IK-1) = AMOD(OPRND(IK-1), R2)
C     GO TO 806

816 OPRND(IK-1) = OPRND(IK-1)**OPRND(IK)
C     GO TO 806

C 790 IF(ID(INHALT(LNKR(OPRND(IK))))).EQ.7) GO TO 310
C 791 IF(CPRND(IK-1) - OPRND(IK)) 791, 793, 791
C 792 IF(ID(INHALT(LNKR(OPRND(IK))))).NE.0.OR.ID(INHALT(LNKR(OPRND(IK)))) .NE.0) GO TO 8792
C 793 OPRND(IK-1) = 1.0
C     GO TO 792

C 860 IF(I(ICHR(I(INHALT(LNKR(OPRND(IK)))))).EQ.0) GO TO 871
C 861 IF(I(INHALT(LNKR(OPRND(IK))))).EQ.1) GO TO 871
C 862 IF(I(INHALT(LNKR(OPRND(IK))))).EQ.2) GO TO 871
C 863 IF(I(INHALT(LNKR(OPRND(IK))))).EQ.3) GO TO 871
C 864 IF(I(INHALT(LNKR(OPRND(IK))))).EQ.4) GO TO 871
C 866 IF(I(INHALT(LNKR(OPRND(IK))))).EQ.6) GO TO 871
C 867 IF(I(INHALT(LNKR(OPRND(IK))))).EQ.7) GO TO 871
C 868 IF(I(INHALT(LNKR(OPRND(IK))))).EQ.5) GO TO 871
C     GO TO 873

C 871 OPRND(IK) = 1.0
C     ICPRND(IK) = 1
C     GO TO 873

C 873 OPRND(IK) = 0.0
C     IOPRND(IK) = 1
C     GO TO 873

C 817 OPRND(IK) = 1.0
C 818 ICPRND(IK) = 1
C     IK = IK + 1
C     GO TO 10

C 819 OPRND(IK) = 0.0
C     GO TO 818
C 820 IF(CPRND(IK).GT.0.0) GO TO 871
GO TO 873

C 930 IF(IOPRND(IK).NE.2) GO TO 8930
I=LNKR(OPRND(IK))-1
JMPRTN=JMPRTN+LNKL(OPRND(IK))-BLKNUM
IF(JMPRTN.LT.0) JMPRTN=0
IK=0
GO TO 10

C 937 IOPRND(IK)=0
GO TO 10
ITEMP=OPRND(IK)
OPRND(IK)=ITEMP
GO TO 10

939 IF(OPRND(IK).GE.+0.0) GO TO 10
781 IF(IOPRND(IK).NE.0) GO TO 8806
OPRND(IK)=~OPRND(IK)
GO TO 10

8310 WRITE(6,9310)
STOP

8511 WRITE(6,9511)
STOP

8832 WRITE(6,9832)ITEMP
STOP

8930 WRITE(6,9930) I
STOP

8716 WRITE(6,9716) IVRBL
STOP

8845 WRITE(6,9845)CODE(I),CODE(I+1)
STOP

8740 WRITE(6,9740)I
STOP

8745 ITEMPT=INHALT(LNKR(OPRND(IK-1))-1)
WRITE(6,9745)ITEMPT
STOP

8650 WRITE(6,9650)
STOP

8775 WRITE(6,9775)
STOP

8806 WRITE(6,9806)
STOP

8811 WRITE(6,9811)
STOP

8736 WRITE(6,9736)KLBL
STOP

8792 WRITE(6,9792)
STOP

3860 WRITE(6,9860)
STOP

58510 WRITE(6,9510)
STOP

49005 FORMAT(I40,/) 
9006 FORMAT(I40,5X,12A6)
9310 FORMAT(I40,52HATTEMPTED COMPARISON OF CHARACTER AND NONCHARACT
IER. }
9930 FORMAT(37HNO LABEL SUPPLIED TO JUMP INSTRUCTION,5X,A6)
9931 FORMAT(37HILLEGAL COMMAND OR OPERAND. TERMINATE)
9950 FORMAT(33HLIST OPERATION ON NONLIST OPERAND)
9511 FORMAT(20HSUFFIX OF EMPTY LIST)
9650 FORMAT(46HSUBSCRIPT CALLED FOR LIST CELLS NOT YET EXTANT)
9716 FORMAT(12HREFERENCE TO,2X,A6,2X,24HAN UNDECLARED IDENTIFIER)
9736 FORMAT( A6,1X,35HIS NOT USED AS A LABEL IN ITS BLOCK)
9740 FORMAT(8HAT CODE(,I6,35H), ASSIGNMENT OF VALUE TO VALUE. }
9745 FORMAT(45HAASSIGNMENT OF LOCAL REFERENCE TO GLOBAL IDENT,2X,A6)
9775 FORMAT(47HATTEMPTED LOGICAL OPERATION, NONLOGICAL OPERAND)
9792 FORMAT(45HREATION SOUGHT BETWEEN NONNUMERICAL OPERANDS)
9806 FORMAT(43HARITHMETIC ATTEMPT ON NONNUMERICAL OPERANDS)
9811 FORMAT (26HDIVISION BY ZERO ATTEMPTED)
9832 FORMAT(A6,36HLABEL NOT DECLARED AT HEAD OF BLOCK)
9845 FORMAT(38HPARAMETERLESS PROCEDURE CALL EXPECTED.,2X,A6,A6)
9860 FORMAT(48HPROCEDURE CALL ON NONPROCEDURE VARIABLE AND LIST)
END
SUBROUTINE INITAS(M,N)
C
C THIS SUBROUTINE Initializes THE STRUCTURE OF THE DIMENSIONED
C ARRAY M INTO A LIST OF AVAILABLE SPACE WITH READER CELL AVSL.
C DIMENSION M(N)
COMMON/AVSLX,AVSL,CODE(500)
C
DO 2 I = 1,N
M(I) = 0
K = N-2
DO 3 I = 1,K,2
CALL SETDIR(-1,-1,MADOV(M(I+2)),M(I))
CALL SETDIR(-1,MADOV(M(I)),MADOV(M(N-I)),AVSL)
RETURN
END
SUBROUTINE ERASE(LINK)

C
C RETURNS FIRST CELL OF LIST TO LAVS IF REFERENCE COUNT IS ZERO
C OR ONE. OTHERWISE, DECREMENTS REFERENCE COUNT OF THE FIRST CELL.

C
COMMON/AVSLX/AVSL, CODE(500)
ILINK = INHALT(LINK)
IF(LNKL(ILINK).GT.1) GO TO 2060
CALL SETIND(-1,-1,LINK,LNKL(AVSL))
CALL SETDIR(-1,IBTM(LINK,+1),-1,AVSL)
RETURN

2060 CALL SETIND(-1, LNKL(ILINK)-1, -1, LINK)
RETURN
END
FUNCTION IBTM(LLNK, IFLAG)

C IF IFLAG=-1, FINDS THE LAST CELL OF A NONEMPTY LIST WHOSE FIRST
C CELL HAS ADDRESS GIVEN BY LLNK. IF IFLAG=+1, RETURNS THE LAST LIST
C CELL NOT REFERENCED BY ANOTHER LIST NAME.

C

LLNR=LLNK
LLNN=LLNR
LLNR=LNKR(INHALT(LLNN))
IF(IFLAG,3000,2001
2001 IF(LNK,L(INHALT(LLNR)))3000,2002,2005
2002 IF(LLNR)3000,2005,2000
2005 IBTM=LLNN
RETURN
3000 WRITE(6,3005)
3005 FORMAT(1HO,14HERORD IN IBTM.
STOP
ENC
FUNCTION NUCELL(X)

C GETS A NEW CELL FROM LAVS, IF THAT CELL HAS A SUBLIST, C
C RETURNS THE SUBLIST TO LAVS BY CALLING ERASE.
C
COMMCA/AVSLX/AVSL, CODE(500)
M=LNKR(AVSL)
IF(M.GT.0) GO TO TC 1
2 WRITE(6,901)
STOP
1
M=M=INHALT(M)
IF(ID(MM).NE.3.OR.LNKR(INHALT(M+1)).EQ.0) GO TO TC 3
4 CALL ERASE(LNKR(INHALT(M+1)))
3 CALL SETDIR(-1,-1,LNKR(MM),AVSL)
CALL STRIND(0,M)
CALL STRIND(0,M+1)
NUCELL = M
RETURN
901 FORMAT(1h1, 6x, 33HLAVS EXHAUSTED-NUCELL TERMINATION)
END
SUBROUTINE COPY(LLNK,MTOP,MBOT)

C
C Puts zero into LNKL of all cells of copied list. If LLNK = 0,
C an empty cell is returned. If LNKL (INHALT(LLNK)) = 0, returns
C LLNK.

COMPON/AVSLX/AVSL, CODE(500)
MTOP = LNKR(AVSL)
KTOP = LLNK
IF(KTOP.EQ.0) GO TO 2051
IF(LNKL(INHALT(KTOP)).GT.0) GO TO 205C
MTOP = KTOP
MBCT = IBTM(MTOL,-1)
RETURN

2050 IKTOP = INHALT(KTOP)
IKTOP1 = INHALT(KTOP+1)
JTOP = NUCELL(DUMMY)
CALL SETIND(IC(IKTOP),0,LNKR(AVSL),JTOP)
CALL STRIND(IKTOP1,JTOP+1)
IF(ID(IKTOP1).NE.3.OR.LNKR(IKTOP1).LE.0) GO TO 2052
CALL SETIND(-1,LNKL(IKTOP)+1,-1,IKTOP1)

2052 MBOT = LNKR(IKTOP)
IF(MBCT.LE.0) GO TO 2054
KTOP = MBOT
GO TO 2050

2054 MBOT = KTOP
RETURN

2051 MBOT = NUCELL(DUMMY)
CALL SETIND(3,0,0,MBOT)
RETURN

END
SUBROUTINE OUTOUT(I TYPE, CONTENT, LOCAT)
DATA IFLSE, ITRUE, Ilabel, IUNDF, IREFRN, IPROC0D/
1 5HFLSE, 4HTRUE, 5Hlabel, 6HUNDF, 6HREFRN, 6HPROC0D/
LOCAT=LOCAT+1
1000 IF(LOCAT.LE.12) GO TO 2000
LOCAT=1
WRITE(6,1050)
1050 FCRMAT(1H, 80X)
2000 IF(I TYPE.NE.0) GO TO 2070
GO TO 3900
2070 IF(I TYPE.NE.1) GO TO 2071
IF(CONTENT.EQ.0.0) ICONTN=IFLSE
IF(CONTENT.EQ.1.0) ICONTN=ITRUE
GO TO 2390
2071 IF(I TYPE.NE.2) GO TO 2072
ICONTN=IUNDF
GO TO 2990
2072 IF(I TYPE.NE.4) GO TO 2073
ICONTN=ITRUE
GO TO 2900
2073 IF(I TYPE.NE.5) GO TO 2074
ICONTN=IREFRN
GO TO 2900
3000 GO TO (51, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512), LOCAT
401 WRITE(6, 3401) ICNTN
RETURN
402 WRITE(6, 3402) ICNTN
RETURN
403 WRITE(6, 3403) ICNTN
RETURN
404 WRITE(6, 3404) ICNTN
RETURN
405 WRITE(6, 3405) ICNTN
RETURN
406 WRITE(6, 3406) ICNTN
RETURN
407 WRITE(6, 3407) ICNTN
RETURN
408 WRITE(6, 3408) ICNTN
RETURN
409 WRITE(6, 3409) ICNTN
RETURN
410 WRITE(6, 3410) ICNTN
RETURN
411 WRITE(6, 3411) ICNTN
RETURN
412 WRITE(6, 3412) ICNTN
RETURN
501 WRITE(6, 3501) CONTENT
<table>
<thead>
<tr>
<th>SCHNEIDER, VI</th>
<th>OUTLINX</th>
<th>001/68/799</th>
<th>EFN</th>
<th>SOURCE STATEMENT</th>
<th>IFN(S)</th>
<th>DATE 01/0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GC TO 2076</td>
<td>WRITE(6,3502)CONTNT</td>
<td>GC TO 2076</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GC TO 2076</td>
<td>WRITE(6,3503)CONTNT</td>
<td>GC TO 2076</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GC TO 2076</td>
<td>WRITE(6,3504)CONTNT</td>
<td>GC TO 2076</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GC TO 2076</td>
<td>WRITE(6,3505)CONTNT</td>
<td>GC TO 2076</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GC TO 2076</td>
<td>WRITE(6,3506)CONTNT</td>
<td>GC TO 2076</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GC TO 2076</td>
<td>WRITE(6,3507)CONTNT</td>
<td>GC TO 2076</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GC TO 2076</td>
<td>WRITE(6,3508)CONTNT</td>
<td>GC TO 2076</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GC TO 2076</td>
<td>WRITE(6,3509)CONTNT</td>
<td>GC TO 2076</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GC TO 2076</td>
<td>WRITE(6,3510)CONTNT</td>
<td>GC TO 2076</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GC TO 2076</td>
<td>WRITE(6,3511)CONTNT</td>
<td>GC TO 2076</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GC TO 2076</td>
<td>WRITE(6,3512)CONTNT</td>
<td>GC TO 2076</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOCAT=LOCAT+1</td>
<td>RETURN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3401</td>
<td>FCRMAT(1H+,A6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3402</td>
<td>FCRMAT(1H+,6X,A6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3403</td>
<td>FCRMAT(1H+, 12X,A6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3404</td>
<td>FORMAT(1H+,18X,A6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3405</td>
<td>FORMAT(1H+,24X,A6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3406</td>
<td>FORMAT(1H+,30X,A6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3407</td>
<td>FCRMAT(1H+,36X,A6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3408</td>
<td>FORMAT(1H+,42X,A6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3409</td>
<td>FORMAT(1H+,48X,A6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3410</td>
<td>FORMAT(1H+,54X,A6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3411</td>
<td>FCRMAT(1H+,60X,A6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3412</td>
<td>FCRMAT(1H+,66X,A6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3501</td>
<td>FORMAT(1H+,F12.4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3502</td>
<td>FORMAT(1H+,6X,F12.4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3503</td>
<td>FORMAT(1H+,12X,F12.4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3504</td>
<td>FORMAT(1H+,18X,F12.4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3505</td>
<td>FORMAT(1H+,24X,F12.4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3506</td>
<td>FORMAT(1H+,30X,F12.4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3507</td>
<td>FORMAT(1H+,36X,F12.4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3508</td>
<td>FORMAT(1H+,42X,F12.4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3509</td>
<td>FORMAT(1H+,48X,F12.4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3510</td>
<td>FORMAT(1H+,54X,F12.4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3511</td>
<td>FORMAT(1H+,60X,F12.4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3512</td>
<td>FORMAT(1H+,66X,F12.4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

END
ENTRY SETIND

* THIS FUNCTION SETIND(ID, LNKL, LNKR, CELL) STORES ID IN ID FIELD, LNKL IN LNKL FIELD, LNKR IN LNKR FIELD OF CELL WHOSE ADDRESS IS IN WORD NAMED CELL. IF -1 APPEARS AS ANY PARAMETER BUT CELL, THAT FIELD IS LEFT UNCHANGED.

SETIND STZ MM
CLA* 6,4
STA **+9

STA **+15
STA **+17
STA **+18
CLA* 3,4
TMI **+5
ALS 30

STO MM
CAL MASK1
ANS **
CLA* 4,4
TMI **+6
ANA MASK2
ALS 15
ORS MM
CAL MASK3
ANS **

CLA* 5,4
TMI **+2
STA **
CLA MM
ORS **
TRA 1,4

MM BSS 1
MASK1 OCT 007777777777
MASK2 OCT 000000777777
MASK3 OCT 770000077777

END
ENTRY STRIND

* THIS FUNCTION STRIND(DATUM,IADRES) STORES THE VALUE
* NAMED BY DATUM IN THE CELL WHOSE ADDRESS IS NAMED
* BY IADRES.

<table>
<thead>
<tr>
<th>STRIND</th>
<th>CLA*</th>
<th>GET ADDRESS OF CELL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>STA</td>
<td>**+2</td>
<td></td>
</tr>
<tr>
<td>CLA*</td>
<td>3,4</td>
<td>GET DATUM'S VALUE.</td>
</tr>
<tr>
<td>STO</td>
<td>**</td>
<td>STORE IT.</td>
</tr>
<tr>
<td>TRA</td>
<td>1,4</td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ENTRY STRDIR

* THIS FUNCTION STRDIR(DATUM,CELL) STORES THE VALUE
* NAMED BY DATUM IN THE WORD NAMED BY CALL, DATUM
* CAN BE EITHER FIXED OR FLOATING POINT AND THE FUNCTION CAN BE
* NESTED.

<table>
<thead>
<tr>
<th>STRDIR</th>
<th>CLA*</th>
<th>3,4</th>
</tr>
</thead>
<tbody>
<tr>
<td>STO*</td>
<td>4,4</td>
<td></td>
</tr>
<tr>
<td>TRA</td>
<td>1,4</td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

GET DATUM'S VALUE
ENTRY SETDIR

* THIS FUNCTION SETDIR(ID, LNKL, LNKR, CELL) STORES ID IN ID FIELD.

* LNKL IN LNKL FIELD LNKR IN LNKR FIELD OF WORD NAMED CELL.

* IF -1 APPEARS AS ANY PARAMETER BUT CELL, THAT FIELD IS

* LEFT UNCHANGED.

<table>
<thead>
<tr>
<th>SETDIR STZ</th>
<th>MM</th>
<th>INITIALIZE MM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLA* 3,4</td>
<td></td>
<td>GET THE ID</td>
</tr>
<tr>
<td>TMI 5</td>
<td></td>
<td>TEST FOR NEGATIVE</td>
</tr>
<tr>
<td>ALS 30</td>
<td></td>
<td>STORE SHIFTED ID FIELD</td>
</tr>
<tr>
<td>STO MM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAL MASK1</td>
<td></td>
<td>MASK OUT THE ID PORTION</td>
</tr>
<tr>
<td>ANS* 6,4</td>
<td></td>
<td>END</td>
</tr>
<tr>
<td>CLA* 4,4</td>
<td></td>
<td>GET THE LNKL</td>
</tr>
<tr>
<td>TMI 6</td>
<td></td>
<td>TEST FOR NEGATIVE</td>
</tr>
<tr>
<td>ANA MASK2</td>
<td></td>
<td>BLOCK OUT OTHER FIELDS</td>
</tr>
<tr>
<td>ALS 15</td>
<td></td>
<td>SHIFT INTO POSITION</td>
</tr>
<tr>
<td>ORS MM</td>
<td></td>
<td>AND STORE IN TEMPORARY</td>
</tr>
<tr>
<td>CAL MASK3</td>
<td></td>
<td>MASK OUT LNKL</td>
</tr>
<tr>
<td>ANS* 6,4</td>
<td></td>
<td>PORTION OF CELL</td>
</tr>
<tr>
<td>CLA* 5,4</td>
<td></td>
<td>GET THE LNKR</td>
</tr>
<tr>
<td>TMI 7</td>
<td></td>
<td>TEST FOR NEGATIVE</td>
</tr>
<tr>
<td>STA* 6,4</td>
<td></td>
<td>OVERLAY LNKR OF CELL</td>
</tr>
<tr>
<td>CLA MM</td>
<td></td>
<td>FETCH THE TEMPORARY</td>
</tr>
<tr>
<td>ORS* 6,4</td>
<td></td>
<td>STORE THE TEMPORARY</td>
</tr>
</tbody>
</table>

| TRA 1,4   |    | EULER |
| MM BSS 1  |    |     |
| MASK1 OCT | 0077777777777 | EULER |
| MASK2 OCT | 0000007777777 | EULER |
| MASK3 OCT | 7700000777777 | EULER |
| END       |    | EULER |
ENTRY CONT
ENTRY INHALT

* DELIVERS THE CONTENTS OF THE WORD WHOSE MACHINE ADDRESS
* IS THE PARAMETER, AND IS STORED AS AN INTEGER. CONT IS USED
* IN FLOATING POINT TO PREVENT TYPE CONVERSION AND INHALT IS
* USED FOR INTEGER ARITHMETIC TO FOOL THE SYSTEM.

<table>
<thead>
<tr>
<th>CONT</th>
<th>TRA</th>
<th>INHALT</th>
<th>CLA*</th>
<th>STA</th>
<th>**</th>
<th>CLA</th>
<th>**</th>
<th>TRA</th>
<th>**</th>
<th>GETS ADDRESS STORED IN PARAMETER.</th>
<th>END</th>
</tr>
</thead>
<tbody>
<tr>
<td>**+1</td>
<td></td>
<td>3,4</td>
<td></td>
<td>**1</td>
<td></td>
<td></td>
<td></td>
<td>1,4</td>
<td></td>
<td>EULER</td>
<td>EULER</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>EULER</td>
<td>EULER</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>EULER</td>
<td>EULER</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>EULER</td>
<td>EULER</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>EULER</td>
<td>EULER</td>
</tr>
</tbody>
</table>
IDX 0001

**ENTRY ID**

* THIS PRIMITIVE FUNCTION RETURNS AS AN INTEGER THE ID FIELD
* OF THE CELL NAMED AS PARAMETER.

<table>
<thead>
<tr>
<th>ID</th>
<th>CLA*</th>
<th>3,4</th>
<th>GET THE CELL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARS</td>
<td>30</td>
<td></td>
<td>EULER</td>
</tr>
<tr>
<td>TBA</td>
<td>1,4</td>
<td></td>
<td>EULER</td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
<td>EULER</td>
</tr>
</tbody>
</table>

EULER
**LNKLO001**

**ENTRY LNKL**

* This primitive function returns as an integer the LNKL field of the cell named as a parameter.

<table>
<thead>
<tr>
<th>LNKL</th>
<th>CLA*</th>
<th>3,4</th>
<th>GET CELL.</th>
<th>EULEK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANA</td>
<td>MASK</td>
<td></td>
<td>MASK OUT ID AND LNKR</td>
<td>EULER</td>
</tr>
<tr>
<td>ARS</td>
<td>15</td>
<td></td>
<td></td>
<td>EULER</td>
</tr>
<tr>
<td>TRA</td>
<td>1,4</td>
<td></td>
<td></td>
<td>EULER</td>
</tr>
<tr>
<td>MASK</td>
<td>OCT</td>
<td>007777700000</td>
<td>5-7'S FOLLOWED BY 5-0'S</td>
<td>EULER</td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
<td></td>
<td>EULER</td>
</tr>
</tbody>
</table>
**ENTRY LNKR**

* THIS PRIMITIVE FUNCTION PRESENTS AS AN INTEGER THE MACHINE ADDRESS CONTAINED IN THE RIGHT LINK FIELD OF THE CELL NAMED.

<table>
<thead>
<tr>
<th>LNKR</th>
<th>CLA*</th>
<th>GET CELL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3,4</td>
<td></td>
</tr>
</tbody>
</table>

**ANA MASK**

MASK OUT ID AND LNKL

| TRA | 1,4 |

**MASK OCT**

000000077777 70'S AND 5-7'S.

**END**
ENTRY MADOV

* FETCHES THE MACHINE ADDRESS OF THE CELL NAMED AS
* A PARAMETER

MADOV CLA 3,4 GET THE LOCATION OF THE CELL.
TRA 1,4 EULED
END EULED
ENTRY  JUSTIN  

JUSTIN(FL,A,B)=THE BCD NUMBER STORED IN A AND B IS CONVERTED TO FL.

<table>
<thead>
<tr>
<th>JUSTIN</th>
<th>SXA</th>
<th>BACK,1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PXA</td>
<td>0,0</td>
</tr>
<tr>
<td></td>
<td>LDQ*</td>
<td>4,4</td>
</tr>
<tr>
<td></td>
<td>AXT</td>
<td>6,1</td>
</tr>
<tr>
<td></td>
<td>RQL</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>LGL</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>TIX</td>
<td>**-2,1,1</td>
</tr>
<tr>
<td></td>
<td>LDQ*</td>
<td>5,4</td>
</tr>
<tr>
<td></td>
<td>AXT</td>
<td>6,1</td>
</tr>
<tr>
<td></td>
<td>RQL</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>LGL</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>TIX</td>
<td>**-2,1,1</td>
</tr>
<tr>
<td></td>
<td>SLW*</td>
<td>3,4</td>
</tr>
<tr>
<td>BACK</td>
<td>AXT</td>
<td>**,1</td>
</tr>
<tr>
<td></td>
<td>TRA</td>
<td>1,4</td>
</tr>
<tr>
<td></td>
<td>END</td>
<td></td>
</tr>
</tbody>
</table>
ISHF0001

* SHIFTS THE ARGUMENT RIGHT ONE CHARACTER AND RETURNS VALUE
ENTRY ISHIFT

ISHIFT CAL* 3,4
ARS 6
TRA 1,4
END
INTRSC COMPARES TWO ALPHANUMERIC WORDS, RETURNING 0 IF THEY MATCH

<table>
<thead>
<tr>
<th>INTRSC</th>
<th>CAL*</th>
<th>3, 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERA*</td>
<td>4, 4</td>
<td></td>
</tr>
<tr>
<td>TRA</td>
<td>1, 4</td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

COMPLEMENT OF LOGICAL WORD
PUT IN AC

EULDE
30. FIOS  33715
   .FBDBF  33621  .EVEN  33633  .DDDFL  33654
   .MOD  33657  .PEX  3366C  .FXP  33661
   .FIOS  33715  .FSEL  3407C  .FILR  34074
   .FILL  34113  .FCLS  34115  .FOPN  34121
   .REED  34276  .BIN  34277  .FCT  34300
   .TAP7  34361
31. FIOH  34366
   .FIOH  34366  .FFIL  35227  .FRTN  35255
32. FWRD  35445
   .FWRD  35445
33. FRDD  35655
   .FRDD  35655
34. UN05  35703
   .UN05  35703
35. UN06  35704
   .UN06  35704  .BUFSZ  35705
36. FLOG  35710
   ALOG10  35710  ALOG  35711
37. FXPF  36114
   EXP  36114
38. FXP3  36235
   XP3  36235
39. IOCS  36362
   .L(I0)  36362  .MONSW  36402  .TEOR  36451
   .CLOSE  36614  .ATT1C  36627  .SH1  37041
   .OP4  37152  .UP7  37203  .OP9.2  37217
   .READ  37272  .RER1  37315  .WRT1  37317
   .NEEIT  37640  .GTX  37661  .RW7  37777
   .SEL59  41065  .BSR  41504  .EOTOF  41631
   .TCHEX  42173  .BASIO  42176
40. IOCSM  42201

I/O BUFFERS

42201 THRU 76705

UNUSED CORE

76706 THRU 77014

.BGIN..BGIN.(  *  THIS I,  *  S A DE,  *  MONSTR,
   *  ATION  *  PROGRA,  *  M THAT,  *  HAS,
   *  BEEN T,  *  RANSLA,  *  TED BY,  *  THE E,
   *  ULER  *  SYNTAC,  *  TIC TR,  *  ANSLAT,
   *  CR AND,  *  IS BE,  *  ING  *  INTERP,
   *  RETEC  *  BY THE,  *  MODIF,
   *  IED PO,  *  LISH  *  STRING  *  INTER,
   *  PRETOR,  *  *  *  )  *  OUT  END  *  BGIN..NEW. X

   *  NEW  S  VRBL.S  (.  NMBR.202400000000  *  $  BGIN..VRBL.
   X  VRBL.X  .IN.  NMBR.201400000000  =  *  VRBL.S
   VRBL.X  .IN.  )  .IN. END $  $  $ (  *  X=
   VRBL.X  .IN.  )  .OUT $  =  *  VRBL.X
   VRBL.S  NMBR.201400000000  IN  =  *  X=

   )  IN. END  *  BGIN..NEW. A  .NEW R. VRBL.A (*
   NMBR.201400000000  (. NMBR.202400000000  NMBR.202600000000
(THIS IS A DEMONSTRATION PROGRAM THAT HAS BEEN TRANSLATED BY THE EULER SYNTACTIC TRANSLATOR AND IS BEING INTERPRETED BY THE EULER MODIFIED POLISH STRING INTERPRETOR.)

```
BEGIN
BEGIN(*, *, *, *, *, *
    *, *, *, *, *, *
    *, *, *, *, *, *
    *, *, *, *, *, *)
OUT.

END.
BEGIN
NEW
NEW
VRBL
(, NMBR,
    $, $
    , VRBL
    , VRBL
    , NMBR)
IN. = , , , , ,
    , VRBL
    , IN. )
OUT.

( X = 2.0000)

VRBL
, NMBR.) IN.
BEGIN
VRBL
, VRBL
, IN.
NMBR
= , , , , ,
    , VRBL
    , IN. )
OUT.

( X = 3.0000)

\$ END \$ END.
BEGIN
NEW
NEW
VRBL
(, NMBR,
    , NMBR
    , NMBR)
VRBL
IN. )
OUT.

( A = ( 1.0000, 2.0000, 3.0000
    , 4.0000 )

VRBL
, VRBL
, NMBR.) = , , , , ,
    , VRBL
    , IN. )
OUT.

(R = REFERNC)

VRBL
, IN.
NMBR.) IN.
VRBL
IN. )
OUT.

(R(1) = 2.0000)

VRBL
, IN.
NMBR.) IN.
VRBL
IN. )
OUT.

(R(2) = 3.0000)

VRBL
*, *
VRBL
IN.
NMBR.) IN.
VRBL
IN. )
OUT.

(R(1) = 2.0000)

END.
BEGIN
NEW
NEW
VRBL
(, *
    , VRBL
NMBR)
OUT.

(N = 9.0000)

VRBL
, *
$ =
VRBL
OUT.

(FIBGC = PROCDCR)

VRBL
, *
VRBL
IN.
VRBL
*, *
VRBL
LSCL.
VRBL
IN.
VRBL
IN. )
FRML
VRBL
IN.
NMBR
LEQ.
IF.
NOT.
IF.
VRBL.
```
( FIBO( 9.0000, 4.0000, 9.0000, 16.0000 ) = 55.0000 )

END $, BEGIN NEW NEW NEW VRBL..$ = ,, ( *
, VRBL..NMBR= )..OUT.

( I= 1.0000 )

$,, ( * ,, VRBL..( NMBR,, NMBR,, NMBR,).

( A= ( 4.0000, 9.0000, 16.0000 )

,) VRBL..LSCL..( VRBL..VRBL..IN,, )..IN,, VRBL,, VRBL,.

FRML..FRML..BEGIN..VRBL..IN,, VRBL..IN,, VRBL..IN,, NMBR+, = ,, ( *, VRBL,, VRBL,, VRBL,.IN,, VRBL..IN,, )..IN,, $)...OUT.

( X= 4.0000 )

END $, OUT.

( X= 4.0000 )

VRBL..LSCL..( $,, VRBL,).,, FRML..FRML..BEGIN..VRBL..IN,, VRBL..IN,, VRBL..IN,, NMBR+, = ,, ( *, VRBL,, VRBL,, VRBL,.IN,, VRBL..IN,, )..IN,, $)...OUT.

( X= 16.0000 )

END $, OUT.

( X= 16.0000 )

( I= 3.0000 )

END $, 00000

#LEGAL COMMAND CR OPERAND. TERMINATE