ALGOL TO FORTRAN TRANSLATOR

Wayne T. Wilner

June 1967

*This report also submitted as a CS239 report, Spring quarter, 1967.
I Purpose

Altran converts Extended Algol programs into Fortran IV (H-level) source decks. Most frequently used Algol constructs translate into Fortran code which will run immediately; other constructs are partially translated and will produce Fortran compilation errors. The most powerful and hardest to translate constructs are merely copied into the output deck.

This project had as its initial goal the translation of one particular physics application program. While that program has long since been translated, the program philosophy of pragmatism is still around. This report, too, is eminently practical, and possibly because of that, the less readable.
II  Use

Operations

Deck preparation:

card 1: standard No. 1 card
card 2: standard No. 2 card (system = DISKIO)
card 3: ? EXECUTE ALTRAN/1
card 4: ? DATA ALTRAN
(card 6: output options)

last card: last card of program to be translated

Output options

Output options are specified by inserting a card with \'$\$' in column 1 after the first card with \texttt{begin} on it. No other \$-cards may be in the deck. The \'$\$' may be followed by any combination of:

\begin{itemize}
  \item PUNCH - punched output
  \item LIST - printed output
  \item SOURCE - Algol listing
  \item ARITH - prohibits logical-IF statements
  \item 7090 - inserts IBSYS control cards
  \item 360 - uses EBCDIC character set
\end{itemize}

If no \$-card is used, options LIST and SOURCE are invoked.
III Translation Limitations

This section summarizes the foibles of the procedures which are described in the remaining sections. Here you will find out what is left undone by the translator, there you will find the whys and how comes. Rather than assign guilt to procedures individually, we present the limitations as associated with the logical elements of the language.

Characters

The Fortran alphabet is a 48-character subset of Algol's. Some of Algol's special characters can be expanded: "<" becomes the four-character token ",LT.", while others (e.g., the ampersand) can only be flagged.

Special characters in strings and comments are left intact, and no flag is waved.

Constants

Owing to the larger word on the B5500 than either the 7090 or 360, Algol programmers may be used to writing integers up to $2^{39} - 1$ with as many as 12 digits, and reals from $10^{-47}$ to $10^{+68}$. We do not change the value of any constants, nor do we truncate them to the maximum length.

Identifiers

Unless the first six characters of every identifier are unique, mistakes will be made. For example, "social ≠ socialist" becomes "SOCIAL.NE.SOCIAL". It would be possible to systematically mutilate identifiers involved in conflicts of this kind, but haven't man's conflicts caused enough mutilation?

Fortran has over twice as many reserved words (118) as Algol; you are cautioned against using:

ASSIGN  PAUSE  STOP  PUNCH  PRINT  ATAN  SINH
ERF     DLOG10 DSINC  CEXP  CABS  PDUMP  COMMON
AMOD    MAXO   MINO   SNGL   IDINT  DBLE   ALOG
COS     ATANZ  COSH   GAMMA  DSIN  DATAN  CLOG
CSQRT   BLOCK  IABS   MAXI   MINI  ISIGN  REAL
DMAXX   CMPLX  RETURN CALL   ALOG10 TAN    ARSIN
TANH    LOAMA  DCOS   DATANZ CSIN  EXIT   DATA
AINT    AMACO  AMINO  FLOAT  DIM   AIMAO  DMINI
ENTRY   SLITE  SLITET OVERFL DVCHK  COTAN  ARCOS
DLOG    DMOD   DSQRT  CCOS   DUMP  CONJG  INT
AMAXI   AMINX  IFIX   IDIM   DABS  DSIGN

- 3 -
although, again, the translator could be programmed to systematically modify such identifiers.

Naming conflicts may also arise wherever nested blocks are employed. Blocks do not exist in Fortran--and declarations may be placed anywhere, so later redeclaration of an identifier will not produce a new local variable in Fortran. In other words,

```fortran
begin integer i; ...
  begin integer i; ...
  end
end ...
end
```
when translated into Fortran will produce code that uses only one variable named i and not two as in Algol.

One final naming-conflict situation can arise. Subroutine parameter call-by-value is not provided in Fortran, but can be effected by declaring a local variable, initializing it with the actual parameter, and using it in the subroutine instead of the parameter. We use the same name for the local variable in Algol and modify the formal parameter's name by concatenating a "V" in front. This new name is not checked for uniqueness (because it is empirically unique; how pragmatic).

Expressions

Language extensions that depend on the B5500's internal makeup are not translated. They are flagged, with the source characters copied onto the output. These include:

- partial word designators
- concatenate expressions
- conditional expressions

Operators div and mod are infix in Algol, prefix in Fortran. Since the translator reads the input program without backing up, we can only flag them once they are read.

Boolean operators eqv and imp are not defined in Fortran. Rather than insert a definition into the Fortran output, we decided that their use was so infrequent that we could get away with just flagging them.

- 4 -
Declarations

We cannot translate recursive procedures, recursive switches, switch files, switch formats or switch lists. File declarations are ignored (except internally, to properly interpret write-statements).

Save and own are ignored because all variables are, in effect, both save and own.

Array subscripts must be positive integers. Zero subscripts are specifically forbidden. Because of the difficulties in mapping array subscripts meaningfully, the translator assumes all array lower bounds are '1'. Upper bounds may not be expressions—only integers and integer variables are allowed.

Statements

For-clauses may contain only one element and it must be a step-until clause which uses only positive integer variables. This is to conform to Fortran's restrictions on do-loops.

Action labels (end-of-file branching) cannot be used on 7090 Fortran. They are copied into the output, principally because they are expected to be implemented in 360 Fortran.

File actions other than read and write are not translated.

Stream procedures depend on the B5500's word/byte duality. They are not translated.

Double statements depend on stack organization and are not translated.

Zip, wait and when are functions of the B5500 Master Control Program and are not translated.

Blocks

Programs are assumed to have the Fortran structure:

\[
\text{begin}
\begin{align*}
\text{global declarations} \\
\text{procedure p1(...); begin} \\
\text{local variables ... end p1;} \\
\text{...} \\
\text{procedure pN(...); begin} \\
\text{local variables ... end pN;}
\end{align*}
\]
< statement, >;
... < statement$^M$ >
end.

Procedures within procedures are not recognized as such. Common statements and an end statement are generated as if they were on the outermost level. These cards can, naturally, be moved as a unit to a position between two outer-level procedures.

Nested blocks are handled just like nested procedures. Their common statement and end statements can simply be removed.
IV Description

Basic Construction

Design of the translator has been a historic progression from grand oversimplification up to mere oversimplification. In the beginning, the translator consisted of a single stream procedure that reduced the Algol character set to a Fortran character set. Input was copied with a few character-by-character changes such as

```
" - " to "=:
" [ " to "( 
"@" to "***
"=" to ".EQ."
";" to additional card
```

Soon afterwards this procedure was replaced by a family of scanning and emitting procedures which manipulated input according to basic structural parts (identifiers, numbers, reserved words, strings, comments and so on). Each item was transmuted to Fortran form: identifiers--truncated to six characters, labels--converted to numbers, strings--prefixed by nH where n is the number of characters between quote marks, reserved words--changed as required, e.g., 'procedure' to 'subroutine' or 'function', special characters--altered as before.

As soon as an item was scanned, its transmogrification was emitted (placed in the output buffer). Since Algol and Fortran do not map character-to-character, the input and output were usually not coincident. In the worst case, the next-to-last item emitted would be on an output buffer that had been released to the punch (aut al.). For this reason, once an item was emitted it was considered out of the game.

At this stage of development the translator operated on an item-by-item basis. The next sweeping change converted it to statement-by-statement operation. One procedure was restricted to translating the Algol <primary>; another, the <term>; a third, the <arithmetic expression>; and a fourth, the <Boolean expression>. These were then combined to translate assignment statements,

```
<primary>  ←  <arith expr> | <Boole expr>
```

- 7 -
conditional statements,

\[ \text{if} \ <\ \text{Boolean expression} > \ \text{then} \ <\ \text{statements} > \ | \ \text{else} \ <\ \text{statement} > \ |
\]

compound statements,

\[ \text{begin} \ <\ \text{statement} > \ \ldots \ <\ \text{statement} > \ \text{end} \]

and other statements.

The final major overhaul attended to block-by-block translation. In the first pass, all executable statements and procedure bodies were translated. In a second pass declarations were ground out while common statements inserted global variables into each subroutine.
Elements of the Translator

In the description of variables which is to follow, one finds many variables that would have been more logically coded as procedure parameters or local variables. To do so would have been to invite stack overflow. While processing the statement "A ←B", twenty subroutines pile eight levels deep. And this statement might have been inside an expression in a statement of a block controlled by a for-statement of a procedure within a procedure of a block within... ad infinitum.

Variables and data structures

Identifiers used in the program are, in order of appearance,

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G00001</td>
<td>see nul</td>
</tr>
<tr>
<td>next</td>
<td>see scan</td>
</tr>
<tr>
<td>seq</td>
<td>see enew</td>
</tr>
<tr>
<td>rel</td>
<td>see be, ifstmt</td>
</tr>
<tr>
<td>herr ... h12</td>
<td>see stepit</td>
</tr>
<tr>
<td>here ... c12</td>
<td>see scan</td>
</tr>
</tbody>
</table>

info contains the symbol table as a bucket-sorted list of record classes. Row 0 contains the 256 buckets. Row 15 is reserved for buffered access to the rest of info. Information is sequentially stored into the other rows--row 1 is filled before anything appears in row 2.

Record classes exist for identifiers, common variables, and blocks. Identifier records consist of five or more consecutive words in the format:

1. Bucket-list pointer stacklink number of dimensions idarraybits, procedure flag idprobit
2. Identifier up to 6 characters idid
3. block in which declared idblock < 100
4. type declared idtype 1 < 7
5. list pointer to identifier of same type in same block-or-Fortran label to which the Algol label has been equated
6. name of common block to which identifier belongs (if one)-or-first eight characters of string which id defines
7. list pointer to id of same type in same common block (if #6 ≠ 0)-or-second 8 of define
8. first dimension upper bound BCD (if id is an array)-or-third 8
9...higher dimensions, or additional define characters

Words 5 on are also used for storing switch labels, if id is a switch, in order of use.

Common variables records consist of six consecutive words:
1. bucket-list pointer (each common area name is bucket-sorted like a program id)
2. identifier, of the form NNQQ where NN is the number (in BCD) of the interior block in question and QQ the exterior
3. list pointer to real variables in this common block
4. list pointer to integer variables in this common block
5. list pointer to Boolean variables in this common block
6. list pointer to other variables in this common block

Block records consist of five consecutive words:
1. list pointer to labeled common names associated with this block
2. list pointer to real variables declared in this block
3. list pointer to integer variables declared in this block
4. list pointer to Boolean variables declared in this block
5. list pointer to other variables declared in this block

directive used once for a program-generated request to the MCP to remove disk file "partial" "product"
fresh 10 words of blanks, 5 words of zeroes to clear the output buffer
piffle program output buffer--released to printer or punch or both
ntable number of lookahead symbols brought in by the scanner
nlong number of extra-long symbols (> 6 chr) that may be brought in during lookahead
longsize number of words over which any extra-long symbol may extend
table array of lookahead symbols; table [0] = here = where the scanner currently is
longjes array of extra-long symbols for which there wasn't room in table
outsave array for pushdown storage of output buffers (piffle)
oj  'stack' pointer for outsave
punching  true = user wants Fortran cards
listing  true = user wants Fortran listing
source  true = user wants Algol listing
oldif  true = user wants only arithmetic if's
defining  true = input stream is currently in info not from card reader because a defined symbol was encountered
allin  true = "end," was read and card reader has been released
broken  true = string extends across a card boundary
specifications  true = declarations in process refer to formal parameters and not to actual entities
cc  true = user wants IBSYS control cards
029  true = user wants EBCDIC Fortran characters
ext  true = format is explicit and terminates with ")"
unit  see oldifstmt
eamstray  true = inside stream procedure
writedisk  false = enew sends output to printer or punch
true = enew sends output to "partial" "product"
boole  false = arithmetic assignment statement
true = Boolean assignment statement
none  see declare, declar
psym...fpstr  coded result of scanner's latest attempt; contents of this, q.v.
inforowlength  see checkinfo, cutback
infobuffer  value of i when a variable is being entered into the symbol table = $15 \times 256 = 2^{12} - 2^8$
realt...definet  coded type of declaration being scanned; contents of type, q.v.
ids...lists  coded type of items in a declaration list; contents of species, q.v.
linkb  subfield of info words where list pointers are stored
rb  list pointer subfield giving row in info pointed to
nb  list pointer subfield giving word in info row pointed to
infoin  next available info word
infoblock index  first word of current block's info entry
infocom index  first word of current common block's info entry
info buf  first word of reserved area for entering words into info
blockitem  see bc
blockindex see blockstack
less than ... fat special characters in the form in which the scanner handles them
definew ... impw reserved words a la Altran's scanner
skip number of columns to change indentation when entering or leaving a block
idprobit see enlist
idarraybits see enlist
word, chr see typewrite
altran input file, two 10-word buffers
outfil printer output
p punch output
partial disk temporary storage
bc...bo five consecutive words (in the prt) used to buffer block entries into info
clink...co six consecutive words (in the prt) used to buffer common block entries into info
stacklink...idbounds mnemonics for areas in infobuf, see info
assured see checkinfo
blockcount current block number
next block highest block number
comindex see infocomindex
dix see scan
gi see scan
j, k temporary integers
i pointer to info entry of the identifier being scanned (outside of declarations); part of stacklink...idbounds
input number of cards read
total number of cards generated
bad number of cards flagged for errors
margin see inject
nextinfo pointer to next available info word
place see inject
column see scan
this see scan
nu

\ell

sc

lonpreviouscard

dim

w, c

waste

blockstack

savec

ttable

seconds

comingattractions

where

nooj

clockphaseone

cards

whichway

m

n

d

xm

see scan

see scan

see scan

see scan

see awry, enlist

see inject

see scan

array of pointers to block entries, indexed by blockcount

array of types of lookahead symbols; ttable [0] = this

= type of symbol where the scanner currently is

keeps processor time for all Altran phases

see gotlost

see gotlost

see savout

used to read out timer when all cards have been read

see stepit

Altran's error messages; see gotlost

see declar

see declar

see declar, declar

see declar
Procedures and processes

The procedure descriptions which follow are intended to illuminate the many interactions of procedures with each other. Most interactions are accomplished by processing global variables rather than procedure parameters, a syndrome known as side effects. Indeed, the entire translation is, strictly speaking, a side effect--as the main program consists of a single, parameterless procedure call.

Procedures used in the program are, in order of their appearance,

date prints: time of day, month/day/year
move (n, s, d) faster than "for i = 0 step 1 until n do d[i] <- s[i]"
and able to transfer 8-chr data without causing a flat bit interrupt
mkabs (core) converts descriptor to absolute address for convenience of scanner--purely a speed consideration
movestring (slong, s8, s, start, dlong, d8, d, dest, nlong, n8, n)
completely general send; see inject.
send (m, this, there, byte) moves characters under three assumptions:
1. source string begins in character position 2 of its first word:

   0  n A A A · · · 
   0  1  2  3  4  5  6  7

2. destination streamer begins less than 64 characters away from given address, there; byte < 64.
3. character streamer < 64 characters.
Send places nearly everything into the output buffer, piffle; see inject.

zot (icon, altar) loads one character from an alpha constant into the zeroth character of the destination word
places '%' into column 73 of the input image to stop the scanner; see changecards
places 'c' into column 1 of the output buffer for comments; see ecomment, commode
places flags into column 1 of the output buffer; see mark, then enew

passchar

(sc, n) see commode

mark

(a) loads a flag a into word 9 of the output buffer before it gets written onto the disk; later the flag will be zotted into column 1 of the finished image.

controlcard

(t, here) called when scanner lookahead (stepit) finds '$'. It makes continuing calls to stepit (without moving table) looking for special identifiers until it finds '%', end-of-card marker. All global Booleans relating to the manner of user output are reset, to be set when the correct identifier is placed in herr by stepit. If no manner of output is specified (probably by a '$ card' card in the deck), a Fortran listing is given.

scn

(x, s, void, tax, thin, thistax, escape, class, bump) called once for every basic item in the input stream. Given the address at which it last scanned, (escape), the length of the item scanned (including the number of preceding blanks) (bump), the class of item scanned (symbol, number, etc.) (class), scn returns the next item (x), updates the scanning address (s), gives: the number of blanks preceding it (void), the class (symbol, number, etc.) (tax), and the length (thin).

If a number or identifier or string is interrupted by a '%', it may be split across a card boundary. Separate classes for each of these situations are included (pnum, psym, pstr) so that scn may resume on the same item if necessary. A further complication is that strings may include '%' legitimately so an additional class (fpstr) provides the ability to overcome this intricacy.

The code is of interest only to stream procedure devotees.
Scn is destroyed by Stanford octal constants, ""<octal number>"", a mess because of: card boundaries which may appear among the quotemarks; confusion with """"; keeping column straight; and not least because there's no point in trying to put a 48-bit constant into a 36- or 32-bit word.

okcomment (s, d, r, sv, stop) moves 36 characters or less from the input buffer to the output buffer, returning the number moved, setting the Boolean stop to true if it stopped because of a ';' in the input, and updating the scanning address. It provides a 1-1 mapping of comments into the output. see commode.

readback echoes the input onto the output file, centered, with the sequence field set off by '.' and three blanks, just as the Algol compiler lists its source decks. see changecards.

changepcards initializes the scanner on a fresh input buffer, which it also obtains by a release statement. Use of release prohibits input from disk--bad coding!

'%' is put into column 73 (character 0 of the 10th input buffer word) to stop the scanner. Global scanning address sc is set to the absolute address of the input buffer (which is non-overlayable). Global length l and number of preceding blanks w, which add to give sen's bump, are initially zero so that sc is not skipped past column one on entry to sen.

stepit (this, here) lower level scanning administrator--drives sen. Three ways recursive:

1. by an internal transfer ... If a string interrupted by '%' is not split over a card boundary, sen can be re-entered without error if this is set to fpstr; ... If the word 'comment' is seen, commode takes control of the input file until it passes a '!''. Stepit then goes back to sen (and 'comment' may be next);

2. explicitly with cards incremented on each entry and decremented on each exit ... If an item is split across
a card boundary, it may be split across n boundaries and full recursion is most convenient;

3. implicitly by calling controlcard ... '$' signifies the start of directives for Altran and controlcard calls stepit to read them in.

Besides recognizing 'B', '%' and 'comment,' stepit does arithmetic for scn which is extremely awkward in character mode:

(a) adding up $ and waste and lonpreviouscard for an accurate column count;

(b) modifying $, waste, column and here when quote marks (around strings) are to be passed over; and preceding strings by nu in BCD [the required 'H' was supplied by scn].

see scan

higher-level scanning administrator--drives stepit and steps the table each time it is entered. Originally, here (the symbol being scanned), this (its class), nu (its length), and next (the next symbol to be scanned) were all set by scn directly. Scan was just a parameterless way of calling scn. When define capability was added, scn had to scan info as well as infil and scan was programmed to switch between the two. At the same time, symbol lookahead was added.

Each item scanned by scn is placed in herr, starting in character 2 ([12:6]) and extending for as many words as necessary (actually, all identifiers and strings are assumed to contain at most 90 and 87 characters, respectively. Longer items can cause flag bit problems or other arbitrary fatal system errors).

Herr is moved to the forward end, table [n table], of table, the lookahead array, and shifted down toward the current end, table [0]. Its class is shifted in parallel in table. Items which are longer than six characters are moved instead to a circular stack, longjes, and the negative of its index in
longjes is placed instead in table. Upon reaching table [0], the long item is retrieved and laid into here and succeeding prt words from which it will be emitted. After shifting table and ttable, scan calls stepit to fill the vacated forward end. Stepit, one may recall, doesn't quit until herr is not a '$', '%', 'comment', or something split across a card boundary. There is one exception: the '%' following either a '$' or a comment or series of comments which extend over more than one card. In this case, scan simply transfers back to call stepit anew.

Any symbol which is brought in may be a define. Its record is taken out of info and checked. Defined symbols cause the scanner's (scn's) vital statistics (sc, $, waste) to be saved in a pushdown stack, savec (with stack pointer dix) and reset to the declared string in info. Since info may be overlaid, fatal errors can result inside defines in a multi-processing environment. Later, when stepit returns a '#', the saved sc, $, waste are restored. Once 'end.' has been scanned, stepit is never again activated ... only table shifting occurs.

(i) converts identifiers into an eight-bit hash address. see enlist, load, get.

(q, dq) performs packed decimal addition for the output image sequence field; see enew.

(output, marg) returns true if the output buffer contains valid information. It inserts 'continue' if a label has been placed but no other code generated. see enew.

(p) converts to 360 Fortran characters. see enew.

final station for all Fortran output. During the input phase, finished code is written a line at a time on disk. During the output phase, the disk file is copied onto the printer or punch; cards are sequenced with BCD -- EBCD conversion if requested; error flags are moved from column 80 to column 1.
continue  (s) puts "-" in column 6 of the output buffer for Fortran continuation cards.

savout  stores the current output buffer into a pushdown array, outsave, along with inject's vital statistics, w, c, place. Gives an error message if recursion has overflowed outsave.

restorout  undoes what savout did.

inject  (nu, here). All Fortran output must go through this routine. It uses send to pack the output buffer (piffle) item by item. If an item will not fit in its entirety on the current line, that line is released to disk (enew) and a continuation card is started with the same amount of indentation as before. If the item is an extra-long string and still cannot fit, then indentation is chucked. If the string still cannot fit, it is chopped up into pieces which will fit.

e  emits (calls inject) for the item currently being scanned. Chops the length, nu, of identifiers to 6 if necessary.

emitn  (h) emits a symbol whose length is contained in bits 6 - 11 . . . which is just the way the scanner introduces every item into Altran.

emit  (h, n) used to emit an item other than the one currently being scanned.

ecomment  used to emit symbols which follow end's in the Algol program; see remark.

commode  drives okcomment to reproduce comments exactly as they occur. Commode is called by stepit when it sees comment at table's forward end. Since the scanner works off the other end of table, the preceding line is probably not complete--it is saved by savout and later restored.
remark

drives ecomment to reproduce comments which follow end and are terminated by end, else, until or ';' (terminator)

flush

gets Altran out of hopeless situations by scanning for the next semicolon; what is passed over is copied, krunched, into the output stream.

flash

(a) flush which will halt if it sees a. Example of use: to recover from multi-element for-lists by finding do and going on.

checkinfo

(n) see enlist.

load

(x) retrieves common records from info, transferring them to clink and five consecutive prt locations so they can be accessed without the double-indexing that info requires. see putincommon.

get

(x) locates identifier records on the bucket lists in info; sets i to the first record word, stacklink. see globalcheck.

deckname

(id) see pro.

gensyn

(g) produces unique Fortran labels: 10, 20, 30, ...

nul

calls gensyn for a new label

labelled

(p) see emitl

emitl

(£) puts label $l$ in columns 1 - 5. If the output buffer already contains a label, it is released and $l$ is put on the next line.

findl

(£) uses get to set i to the info record of Algol identifier $l$; returns the Fortran label to which $l$ has been equated, after writing $l$ out as a comment so that its mnemonic value is carried over into the Fortran.

numberl

see stmt.

scar

brackets the symbol about to be emitted with flags "###" so they can be easily coded out of the Fortran; for instance, the concatenation operator '&amp;' is so rendered.
gotlost  (n) prints error message n; called all over.

globalcheck retrieves the info record of the currently scanned symbol to see if:
1. it is an identifier
2. it was declared in an exterior block
and if so, calls putincommon

terminator (here) sentence terminators are ';', end, else, until

number called by primary to syntactically translate

\((<\text{integer}>) (., <\text{unsigned integer}>)\) (@<\text{integer}>)

primary called to translate: parenthesized arithmetic expressions (which may be followed by a partial word designator); conditional arithmetic expressions; identifiers; logical constants (true \(\rightarrow\) true.) subscripted identifiers; concatenate expressions; functions (arctan \(\rightarrow\) atan); partial word designators; strings; and signed arithmetic expressions or it gives up and goes to the next special character.

term called by ae to translate arithmetic expressions composed of Algol primaries and the operators x, /, *, MOD, +, : =, & as well as stream constructs such as N LIT "."

operator (here) true \(\equiv\) here is +, -, x, /, *, ., & , :=

be (true to tog) translates Boolean expressions either positively or negatively, logically or arithmetically. Arithmetic Boolean expressions are allowed to have only one clause based on a relational operator.

ae translates expressions of terms connected by + or - (see term). ae (arithmetic expression) seems to be called by every procedure in the translator.

elist translates lists, parameter lists, fill statement lists, i/o statement lists, by calling ae again and again as long as it winds up looking at a ,. For-clauses found in i/o statements are translated by eiofor.
write-list for-clauses presented a horrible problem: in Fortran, all the iteration control follows the list. Thus its translation had to be held until the list itself had been translated. Of course, recursion also reared its ugly head (ugly to Fortran). We chose the most straightforward course: instead of emitting the translated for-clause, we packed its translation into a local array implied. Recursion was trivially easy, since the list was translated by the only procedure that can call eiofor, elist. Error recovery is tricky but as good as can be for most unexpected situations. At the end, implied is emitted by a call to inject.

(species, type) handles the array declaration in Algol. Each variable has to have its bounds declared immediately beside it in Fortran because there is no array declaration. To retrieve the bounds from the end of an <array identifier list>, awry and enlist call each other recursively, saving partial info records (from infobuf) until the bounds are reached. The untwining recursion automatically places the bounds into each identifier’s record.

O, the joys of stack machines!

an atavism. Before ae was written, est (emit statement) was called to translate the entire program. What nonsense!

Now, only fillstmt calls est, because

(i) there may be long strings which used to be truncated in earlier versions of ae and primary

(ii) it is faster than going through the elist-ae-term-primary descent for each list element.

pure poetry. Entered when the scanner is at the '←'. Awe-inspiring.

a fallible way of recovering a <primary> once it has been emitted. By these means, we can emit the primary twice when it is the left part of an assignment within an arithmetic
expression, once in \texttt{inassign}, and again in turn (although, actually, the latter has already happened, before the '=' was seen).

Alternatively, we could perform lookahead in \texttt{table} to anticipate such assignments, but the indefinitely long primary (e.g., 200 subscripts) would require and indefinitely long (fallibly finite) \texttt{table} [and take more time]. Our choice was an expression of taste; it makes \texttt{inassign} elegant.

\textbf{inassign} uses \texttt{pastprimary} to recover the left part of an assignment within an arithmetic expression, saves the output line by \texttt{savout}, calls \texttt{assignstmt} (elegance!) and finally \texttt{restorout}.

\textbf{ecall} translates subroutine calls by: (i) inserting the obnoxious reserved word 'call', (ii) emitting the subroutine name, (iii) handling the parentheses around arguments which \texttt{elist} will handle.

\textbf{oldifstmt} emits the proper sequence of three labels following the arithmetic expression in arithmetic if statements by considering the relational operator \texttt{rel} found by \texttt{be}.

\textbf{ifstmt} changes 'if <be> then <s_1> else <s_2>' into:

\begin{verbatim}
    if . not. <be> go to f    (or if (<ae>) t, f, t)
    <s_1>
    go to j
    f <s_2>
    j continue
\end{verbatim}

using \texttt{nul} to create the artificial labels, \texttt{emitl} to place them, \texttt{be} for the condition, \texttt{stmt} (recursively) for \texttt{s_1} and \texttt{s_2}, and \texttt{enew} to make many lines.

\textbf{forstmt} changes "for <assign> step <integer> until <primary> do <s>" into:

\begin{verbatim}
    DO n, <assign>, <primary>, <integer>
    <s>
    n continue.
\end{verbatim}
If the for-clause is of any other form, Altran jumps to the do and at least translates the iterated statement and indicates the scope of the do-statement. To be accurate, the for-statement translation should be preceded by the test, since Fortran do-statements execute at least once before testing. In other words,

\[
\begin{align*}
\text{IF} & \text{ < assign}> | \text{assigned variable} | . \text{GT. <primary>}, \\
& \text{GO TO } m \\
\text{DO } & n, \text{ <assign>}, \text{ <primary>}, \text{ <integer> } \\
& <s> \\
& n \text{ CONTINUE} \\
& m \text{ CONTINUE}
\end{align*}
\]

is the correct translation.

Step-while-elements ought to be translated from:

"for <left part> ← <ae> step <ae> while <be> do <s> " into:

\[
\begin{align*}
n & \text{ IF } <\text{be}> \text{ GO TO } m \\
<\text{left part}> & ← <\text{ae}> \\
<s> & <\text{left part}> ← <\text{left part}> + (<\text{ae}>) \\
& \text{GO TO } n \\
m & \text{ CONTINUE}
\end{align*}
\]

dostmt
changes "do <s> until <be>" into:

\[
\begin{align*}
m & <s> \\
& \text{if . not . } <\text{be}>, \text{ go to m (or if } <\text{ae}> n, m, n) \\
& \text{n continue}
\end{align*}
\]

whilestmt
changes "while <be> do <s>" into:

\[
\begin{align*}
\ell & \text{ if ( . not . } <\text{be}>) \text{ go to } n \text{ (or if } <\text{ae}> m, n, m) \\
(m) & <s> \\
& \text{go to } \ell \\
& \text{n continue}
\end{align*}
\]
fillstmt changes "fill array row with list" into:

\[
\text{F DATA } \left( \text{array } (\text{III}) \ (\text{III} = 1, *) \right) / <\text{list}> /
\]

where the upper limit on III is left as "*" because we don't know how many items there will be in the list.

gotosstmt translates simple goto's by using findl to get the equivalent Fortran statement number for the Algol symbol.

Switches change from "go to <id> [n]" into go to (p, q, ..., r) where p ... r are statement numbers corresponding to the symbolic labels in the switch declaration.

neststmt handles the strange stream procedure construct

" <stream simple variable> ( <s_1> ... <s_n> ) "

without getting lost.

jumpstmt handles the stream jump statement:

jump out ( <integer> ) (to <label> )

espacing converts write-statement carriage control into Fortran's column 1 carriage control.

\[
[\text{dbl}] \text{ becomes }
\]

\[
\text{n format (1H0)}
\]

\[
\text{write (6, n)}
\]

eformat emits an Algol format as a Fortran one, and its easier than you'd think. First we set the level of parenthesization to zero, put out the Fortran statement number f, up the indentation counter, put out "format" ("), and copy the format item by item until we get to "]" for anonymous formats or")" for declared ones. Only the "Xw" editing phrase needs alteration; in Fortran, it is "wX".

Since carriage control is performed after printing in Algol, we append a "/" or "/1H1" to the format if the write-statement from which it came had [dbl] or [page]. Even though espacing also handles carriage control, writestmt can call only one of the two for any one statement.
writestmt

(writ) tries to cope with:

\[
\begin{align*}
\text{read} & \{ \langle \text{file name} \rangle \} \{ [ \langle \text{carriage control} \rangle ] \} \\
\text{write} & \{ \} \\
\text{empty} & \\
\langle \text{format identifier} \rangle, \\
\langle " " \langle \text{format} \rangle " " \rangle, \\
/, \\
\{ [ \langle \text{action labels} \rangle ] \} \\
\langle \text{arithmetic expression} \rangle, \\
* \\
\end{align*}
\]

The first element may be a file, carriage control, format, " " " "","","","", arithmetic expression, or ")".

Thanks to get we can identify files and formats. If there is no format (default = true), we use statement label 00000 in the output and flag it with "*".

Form is true if a file name was found. No attempt is made to use Fortran file numbers (other than 5 for input (writ = false) and 6 for output).

Label fast is skipped to if there is no list to be written (stuff = false).

rackstring

puts the business end of a define declaration into the identifier record (beginning with the sixth word). Since all blanks are removed in the scanning process, they need to be inserted whenever symbols or numbers follow on symbols. Boolean b remembers symbols for this purpose.

The terminating "#" is sent into info as well because the string will have to be rescanned every time the defined identifier is used.

Rackstring returns the word length of the define.

decl

compares the currently scanned symbol, here, with all possible initial words of declarations. "v" stands for "then if here ≠ " because it looks like the logical "or" (but it has the sense of "nor").
enlist (species, type) is responsible for obtaining all the information that belongs in an identifier's info record for correctly cross-referencing the record and placing it into info. (Strictly speaking, it's never outside of info but we prefer to think of info [15,*] as a separate buffer area; it cannot be, as long as i is to be used to index records whether they are enlisted (placed within info) or not (still in the 'buffer').)

Enlist is called by declarations when the scanner's current item (here) is an identifier in an identifier list of a declaration.

Seemingly unimportant but crucial is the initial setting of i to infobuffer. Many of the symbols in enlist are define's which use i implicitly:

\[ \text{stacklink, idlink, idlab, et al.} \]
\[ \text{info [i . [36:4], i . [40:8]} \]

is the first word in any identifier's record, and enlist corresponds to info [15,0].

We clear stacklink (info [15,0]) which connects the record to the bucket sorted list from which it will be accessed during the rest of the translation.

Idid (info [15,1]) receives the (possibly truncated) name. We generate the name's hash address by scramble and hold it in index. Then the fun begins.

Species, set in declarations, determines which local rearrangements are to be performed on the identifier record. The possibilities are, in numerical order:

- **plain, old variables** - added to the list of identifiers declared in the current block (list pointer is in idlink).

- **arrays** - scanner is moved ahead so upper bounds can be stored (idbounds and up); record is extended by one for each dimension by incrementing length (total = dim); added to id list just like plain, old variables.
labels - equated to a Fortran statement number (idlab).

procedures - a procedure declaration contains code which is immediately translated (thank God for recursion). An unused early bit in stacklink is turned on. Finally, typed procedures are included in the list of identifiers just like plain, old variables.

own variables - This case is never selected. All variables are own in Fortran and all arrays are save.

switches - each label in the declaration is stored from idlab to beyond and the record extended accordingly. see gotostmt.

definitions - the string which constitutes the definition is stored by rackstring, eight bytes per word, starting in idstring (word 6).

formats - assigned a Fortran statement label and emitted by the same routine that handles anonymous formats (with ext = true so that eformat knows to stop at "") "rather than ">").

files - ignored except when needed to distinguish conventional Burroughs write-statements from those indigenous to S. U.

lists - completely ignored, as is the habit at S. U. Better they should be treated like defines, except that hairy local/global considerations may enter.

Once each record is particularized, checkinfo adjusts nextinfo so that the record will not be split across rows of info. Contiguous words are essential for each
record because stream procedures often process records with a single instruction (for instance, move in enlist, get).

Next, the bucket corresponding to the identifier ([info [0, index]]) is made to point to the record (nextinfo) while the old pointer is preserved in stacklink.

After the record is moved from infobuf into info's next available free storage infoin, we adjust the free storage pointer.

Stepping the scanner past the delimiter of the declaration list on which the now-enlisted identifier was found is the final duty of this procedure.

declarations called by block (and as extreme unction by stmt) to admin-
ister enlist to all declared identifiers; decides what type and species to give enlist (e.g., real, array) and advances the scanner to the first identifier.

Make is set false when the first word after a ';' is not one of the 17 reserved words that can start declaration.

vform (x) see pro.

comname (outer, inner) – see putincommon

add the commonblock (new, bc) Each block record in info begins with a pointer to a list of all the labelled common blocks which need to be declared in that block. This list consists of info words linked in their low 12 bits (linkb). Characters 2-5 (comn) contain a labelled common label (e.g., "0802"). Add the compares the first label to new and exits if the label is on the list. Otherwise, it calls itself recursively until the label is found or the end of the list is reduced, where the new label is added.

putincommon is a complicated routine which enters variables into labelled common if they are used in a block interior to the one in which they are declared. See globalcheck. If the variable is not in common, and no common exists for the current block, a common label is generated. ("0802" if the interior block is the eighth block in the program, blockcount, and if the variable was declared in block two, idblock), entered in the symbol table; a record is initialized in clink ... co, after moving the old common record back to info (infocomindex).
Once a common block is available, the variable is added to it, linked by the seventh word in its record (idcomlink), according to type.

If the variable is already allocated to common, whose name is idcom (the sixth word in the identifier record), that common name is used in the final task of putincommon. The common block containing the variable must be added to the list of common for both the inner and outer blocks by addthecommon block.

saveblock
(past) is called whenever a new block is entered (by block and pro). It puts the current block's record into info (infoblockindex), passes the block count back for recursive saving, generates a new blockcount from nextblock and a new blockindex from nextinfo, initializes the block record buffer bc ... bo, and increases the amount of indentation.

cutback
runs through the buckets in the symbol table, disconnecting all variables local to the current block, which is about to be exited. Of course, these variables can still be reached through block and common records.

restoreblock
(past) undoes saveblock's work. It also generates the return and end statements that terminate each subroutine, which will cause grief if there are any nested blocks.

pro
(type) handles procedure declarations, 99% of which concerns the formal parameters—the entire procedure body is elegantly handled by a simple call to stmt.

Since declarations may be contained within this declaration, we declare proitem in which to save the procedure identifier's partial record.

After emitting part of the subroutine/function statement, we store the formal parameter identifiers in formal. Those mentioned in a value specification have bit 2 in their formal word turned on. When these are then emitted, valued parameters receive a "V"-prefix. Later a normal declaration
will be made, followed by an assignment of the form: \(<\text{name}> = V <\text{name}>\). Code within the procedure body will reference \(<\text{name}>\) and not the actual parameter, \(V <\text{name}>\), whose integrity will remain, as in Algol. This concludes the subroutine statement with the side effect of having moved the scanner past the value specification.

A call to \texttt{declarations} takes care of the remaining specifications, with global Boolean \texttt{specifications} = true to inhibit the specification "procedure" from calling \texttt{pro} again. Then we declare the \(V <\text{name}>\) parameters.

By placing \texttt{blockcount} in \texttt{piffle} [12] and writing it to disk by \texttt{enew}, we clue the second phase that common statements and declarations for block number \texttt{blockcount} are to be inserted at that point.

When the \(<\text{name}> = V <\text{name}>\) assignments have been emitted, \texttt{stmt} completes the procedure declaration by translating the body.

Although this is a declaration, by calling \texttt{stmt} we almost certainly will have changed \texttt{i} (by an \texttt{aeprimary-globalecheck-get} chain) and must now repoint it at \texttt{infobuf}.

Lastly, \texttt{eamstray} may have been made true in \texttt{declarations} to enable \texttt{term} to get around unusual \texttt{stream} constructs and is reset before returning to \texttt{enlist}.

\texttt{stmt} alias Superprocedure. After initialization, one call to \texttt{stmt} translates the entire program.

\texttt{Stmt} examines the first one or two items in a statement and calls routines to translate each kind, \texttt{for}, \texttt{do}, \texttt{write}, etc., after translating labels into numeric Fortran labels (by \texttt{number1}).

In order to leave \texttt{stmt} as quickly as possible, we test first for simple assignments--\(<\text{identifier}> \ "\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{...}
if we fail to branch on the words begin, for, go, do, while, we check for "(" after the first word (which characterizes i/o statements and procedure calls). Otherwise we test for the initial item being a subscripted variable or other lengthy primary.

Sometimes the translator really becomes confused, so we look for statements beginning with else, end, ";", or a declaration to avoid a completely bad show. If all is lost, we call flush to get us to a ";", end, else, or until from which we can take some bearing.

`compoundtail` is the second half of `block` which has gobbled up a begin. `compoundtail` calls `stmt` until it sees the machine `end` and uses `remark` to handle comments following that `end`.

`block` calls `saveblock` to create a new block record, calls declarations to handle all declarations, sets `piffle` [12] to `blockcount` so that phase two will insert `common` statements in the right place, calls `compoundtail` to handle all statements, calls `restoreblock` to cut back the symbol table and return to next outer block's record to the block record buffer, print cells be ... bo.

`declar (x£)` emits one identifier within a declaration list. If the list pointer `x£` is not zero, `declar` is called recursively with the next item on the list (so it is the last identifier which is emitted first).

`x£` is actually within an identifier record, the sixth word for variables in `common`, the fourth for all others. To set `i` to the beginning of the record it must be decreased by the global integer `d = 4` or 6 (set by `declare`).

When variables are assigned to common, they are not disconnected from the lists of block in which they are declared. So to avoid double declared. So to avoid double declarations we have to test if we are on a block list (`d = 4`) and if the variable is not in common (`idcom = 0`), or, `d = 4` ⊃ `idcom = 0`.

- 32 -
Global Boolean none keeps us from emitting an empty list or an extra comma at the end of a list. Arrays are dimensioned by emitting the upper bounds that were stored in the identifier record's idbounds ff.

\[\text{declare (xl, dxl)}\]

This drives declar after setting \(d\), the displacement of the \(xl\) list in the identifier records, and initializing none. The typewrite and none statements reset the emitter (inject) and output buffer (piffle) respectively.

\[\text{declarativestatements (blockcount)}\]

Inserts declarations and common statements at the beginning of each block. These are called from lists leading out of the appropriate block record and the attached common records (chained on \(bc\)).

We set the indentation (margin) to what it was when the declarations were encountered, initialize the output buffer, and load the block record indicated by \(\text{blockcount}\) into the block record buffer, \(bc \ldots bo\). Unlike common records, block records are not accessed through the symbol table (info \([0,*]\)); rather, each block is numbered and this number is used as an index in the array blockstack which contains pointers to the first word of each block record in info.

\(bc\) heads a list of common block names, each of which has its record loaded into the labelled common buffer olink \(\ldots co\) and its allocated variables declared; first, in the common statement and again in a specification statement.

\(br, bi\) and \(bb\) head lists of real integer and Boolean variables respectively, local to the block \(\text{blockcount}\).

\[\text{postmortem}\]

Prints some statistics about the translation on the last page.
Index

Variable and procedure identifiers are indexed separately. Parenthesized numbers refer to sequence numbers in the program listing, rather than pages of this report. Indentation signifies that the identifier in question is not explicitly listed, and appears as an ellipsis (...).
<table>
<thead>
<tr>
<th>Variable</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDS...LISTS</td>
<td>11</td>
</tr>
<tr>
<td>INF</td>
<td>12</td>
</tr>
<tr>
<td>INFIL</td>
<td>(0083)</td>
</tr>
<tr>
<td>INFO</td>
<td>9</td>
</tr>
<tr>
<td>INFBUFFER</td>
<td>11</td>
</tr>
<tr>
<td>INFLOOKINDEX</td>
<td>11</td>
</tr>
<tr>
<td>INFBUF</td>
<td>11</td>
</tr>
<tr>
<td>INFOCINDEX</td>
<td>11</td>
</tr>
<tr>
<td>INFJOIN</td>
<td>11</td>
</tr>
<tr>
<td>INFOMIN</td>
<td>11</td>
</tr>
<tr>
<td>INPUF</td>
<td>12</td>
</tr>
<tr>
<td>INTOCOM</td>
<td>(0112)</td>
</tr>
<tr>
<td>IOTA</td>
<td>(0250)</td>
</tr>
<tr>
<td>IS</td>
<td>(0039)</td>
</tr>
<tr>
<td>ISAT</td>
<td>(0039)</td>
</tr>
<tr>
<td>J</td>
<td>12</td>
</tr>
<tr>
<td>K</td>
<td>12</td>
</tr>
<tr>
<td>L</td>
<td>13</td>
</tr>
<tr>
<td>LESSTHAN...FAT</td>
<td>12</td>
</tr>
<tr>
<td>LIFTOL</td>
<td>(0119)</td>
</tr>
<tr>
<td>LINKA</td>
<td>11</td>
</tr>
<tr>
<td>LISTING</td>
<td>11</td>
</tr>
<tr>
<td>LONGJES</td>
<td>10</td>
</tr>
<tr>
<td>LONGSIZE</td>
<td>10</td>
</tr>
<tr>
<td>LONPREVIOUSCARD</td>
<td>13</td>
</tr>
<tr>
<td>M</td>
<td>12</td>
</tr>
<tr>
<td>MARGIN</td>
<td>12</td>
</tr>
<tr>
<td>N</td>
<td>12</td>
</tr>
<tr>
<td>NA</td>
<td>11</td>
</tr>
<tr>
<td>NEXT</td>
<td>9</td>
</tr>
<tr>
<td>NEXTBLOCK</td>
<td>12</td>
</tr>
<tr>
<td>NEXTINFO</td>
<td>12</td>
</tr>
<tr>
<td>NIF</td>
<td>(0090)</td>
</tr>
<tr>
<td>NMR</td>
<td>11</td>
</tr>
<tr>
<td>NPLONG</td>
<td>10</td>
</tr>
<tr>
<td>NONE</td>
<td>11</td>
</tr>
<tr>
<td>NOJO</td>
<td>12</td>
</tr>
<tr>
<td>NTABLE</td>
<td>10</td>
</tr>
<tr>
<td>NU</td>
<td>13</td>
</tr>
<tr>
<td>NUCOM</td>
<td>(0112)</td>
</tr>
<tr>
<td>NUPRIM</td>
<td>(0112)</td>
</tr>
<tr>
<td>NUSTMT</td>
<td>(0112)</td>
</tr>
<tr>
<td>O29</td>
<td>11</td>
</tr>
<tr>
<td>OJ</td>
<td>11</td>
</tr>
<tr>
<td>OLCIF</td>
<td>11</td>
</tr>
<tr>
<td>OUTFIL</td>
<td>12</td>
</tr>
<tr>
<td>OUTSAVE</td>
<td>10</td>
</tr>
<tr>
<td>P</td>
<td>12</td>
</tr>
<tr>
<td>PARTIAL</td>
<td>12</td>
</tr>
<tr>
<td>PIFLIE</td>
<td>10</td>
</tr>
<tr>
<td>PLACE</td>
<td>12</td>
</tr>
<tr>
<td>PAUK</td>
<td>11</td>
</tr>
<tr>
<td>PSTR</td>
<td>11</td>
</tr>
<tr>
<td>PSYM...FPSYM</td>
<td>11</td>
</tr>
<tr>
<td>PUNCHING</td>
<td>11</td>
</tr>
<tr>
<td>RH</td>
<td>REAL</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>DC</td>
<td>SAVEC</td>
</tr>
<tr>
<td>SEG</td>
<td>SKIP</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SUPPRESSEDDIMENSION</td>
<td>SPECIFICATIONS</td>
</tr>
<tr>
<td>SYM</td>
<td></td>
</tr>
<tr>
<td>TABCLEAR</td>
<td></td>
</tr>
<tr>
<td>TABLE</td>
<td></td>
</tr>
<tr>
<td>TABBED</td>
<td></td>
</tr>
<tr>
<td>THIS</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
</tr>
<tr>
<td>TTABLE</td>
<td></td>
</tr>
<tr>
<td>TYPEWRITE</td>
<td></td>
</tr>
<tr>
<td>UNIT</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td></td>
</tr>
<tr>
<td>WASTE</td>
<td></td>
</tr>
<tr>
<td>WHERE</td>
<td></td>
</tr>
<tr>
<td>WHICHWAY</td>
<td></td>
</tr>
<tr>
<td>WORD</td>
<td></td>
</tr>
<tr>
<td>WRI Tedisk</td>
<td></td>
</tr>
<tr>
<td>XM</td>
<td></td>
</tr>
</tbody>
</table>

- 37 -
<table>
<thead>
<tr>
<th>Command</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADTHECOMMONBLOCK</td>
<td>29</td>
</tr>
<tr>
<td>AE</td>
<td>21</td>
</tr>
<tr>
<td>ASSIGNSTMT</td>
<td>22</td>
</tr>
<tr>
<td>AWRY</td>
<td>22</td>
</tr>
<tr>
<td>RE</td>
<td>21</td>
</tr>
<tr>
<td>BLOCK</td>
<td>32</td>
</tr>
<tr>
<td>CHANGECARDS</td>
<td>16</td>
</tr>
<tr>
<td>CHECKINFO</td>
<td>20</td>
</tr>
<tr>
<td>COMMODE</td>
<td>19</td>
</tr>
<tr>
<td>COMNAME</td>
<td>29</td>
</tr>
<tr>
<td>COMPOUNDTAIL</td>
<td>32</td>
</tr>
<tr>
<td>CONTINUE</td>
<td>19</td>
</tr>
<tr>
<td>CONTROLCARD</td>
<td>15</td>
</tr>
<tr>
<td>CUTHACK</td>
<td>30</td>
</tr>
<tr>
<td>DATE</td>
<td>14</td>
</tr>
<tr>
<td>DECKNAME</td>
<td>20</td>
</tr>
<tr>
<td>DECL</td>
<td>26</td>
</tr>
<tr>
<td>DECLAIR</td>
<td>32</td>
</tr>
<tr>
<td>DECLARATIONS</td>
<td>29</td>
</tr>
<tr>
<td>DECLARATIVESTATEMENTS</td>
<td>33</td>
</tr>
<tr>
<td>DECLARF</td>
<td>33</td>
</tr>
<tr>
<td>DOSTMT</td>
<td>24</td>
</tr>
<tr>
<td>E</td>
<td>19</td>
</tr>
<tr>
<td>ECALL</td>
<td>23</td>
</tr>
<tr>
<td>ECOMMENT</td>
<td>19</td>
</tr>
<tr>
<td>EFORMAT</td>
<td>25</td>
</tr>
<tr>
<td>FICFOR</td>
<td>22</td>
</tr>
<tr>
<td>EJECTACARD</td>
<td>18</td>
</tr>
<tr>
<td>ELIST</td>
<td>21</td>
</tr>
<tr>
<td>EMIT</td>
<td>19</td>
</tr>
<tr>
<td>EMITL</td>
<td>20</td>
</tr>
<tr>
<td>EMITN</td>
<td>19</td>
</tr>
<tr>
<td>FNNAME</td>
<td>18</td>
</tr>
<tr>
<td>ENLIST</td>
<td>27</td>
</tr>
<tr>
<td>ESPACING</td>
<td>25</td>
</tr>
<tr>
<td>EST</td>
<td>22</td>
</tr>
<tr>
<td>FILLSTMT</td>
<td>25</td>
</tr>
<tr>
<td>FINDL</td>
<td>20</td>
</tr>
<tr>
<td>FLASH</td>
<td>20</td>
</tr>
<tr>
<td>FLSH</td>
<td>20</td>
</tr>
<tr>
<td>FORSTMT</td>
<td>23</td>
</tr>
<tr>
<td>GENSYM</td>
<td>20</td>
</tr>
<tr>
<td>GET</td>
<td>20</td>
</tr>
<tr>
<td>GLOBALCHECK</td>
<td>21</td>
</tr>
<tr>
<td>GOTO</td>
<td>21</td>
</tr>
<tr>
<td>GOTOSTMT</td>
<td>25</td>
</tr>
<tr>
<td>IFSTMT</td>
<td>23</td>
</tr>
<tr>
<td>INASSIGN</td>
<td>23</td>
</tr>
<tr>
<td>INJECT</td>
<td>19</td>
</tr>
<tr>
<td>INSIDE</td>
<td>16</td>
</tr>
<tr>
<td>JUMPSSTMT</td>
<td>25</td>
</tr>
<tr>
<td>LABELLED</td>
<td>20</td>
</tr>
<tr>
<td>LOAD</td>
<td>20</td>
</tr>
<tr>
<td>MARK</td>
<td>15</td>
</tr>
<tr>
<td>MKABS</td>
<td>14</td>
</tr>
</tbody>
</table>

- 38 -
| MOVE       | 14 |
| MOVESTRING | 14 |
| NESTSTMT   | 25 |
| NUL        | 20 |
| NUMBER     | 21 |
| NUMERL     | 20 |
| 029CHARACTERSET | 18 |
| 0KCOMMENT | 16 |
| GLDIFSTMT  | 23 |
| OPERATOR   | 21 |
| PASTCHAR   | 15 |
| PASTPRIMARY| 22 |
| POSTMORTEM | 33 |
| PRIMARY    | 21 |
| PROC       | 30 |
| PUTINC.COMMON | 29 |
| RACKSTRING | 26 |
| READBACK   | 16 |
| REMARK     | 20 |
| RESTOREBLOCK | 30 |
| RESTOROUT  | 19 |
| SAVERLOCK  | 30 |
| SAVOUT     | 19 |
| SCAN       | 17 |
| SCAR       | 20 |
| SCA        | 15 |
| SCRAMBLE   | 18 |
| SEND       | 14 |
| STEPI      | 16 |
| STMT       | 31 |
| TERM       | 21 |
| TERMINATOR | 21 |
| UP         | 18 |
| VFORM      | 29 |
| WHILESTMT  | 24 |
| WRITESTMT  | 26 |
| ZOT        | 14 |
V. Examples

The first example contains just two untranslatable constructs: array lower bounds of zero, and a read statement without a format. Except for the subscripting problem, the translation would have been evaluated at better than 87%.

The second example used partial word designators, case statements, array rows, stream procedures, and complicated for-lists. Despite these, the translation turned out to be 60% effective.
FOR SOME STRANGE REASON THE SCANNER CHOKES ON THE FIRST "BEGIN"
BEGIN COMMENT CONNECTIVITY ALGORITHMS 
INTEGER I,J,K,M,NJ LABEL STP2J

????? 0 ??? 20 ???, ??? 0 ??? 1 ???
????? USE ONLY POSITIVE INTEGERS (NOT ZERO) FOR ARRAY BOUNDS

INTEGER NOCl BOOLEAN ARRAY COMP[0:20,0:120];

????? 0 ??? 1 ??? 40 ???, ??? 0 ??? 1 ???
????? USE ONLY POSITIVE INTEGERS (NOT ZERO) FOR ARRAY BOUNDS

BOOLEAN ARRAY Sx,SN[0:140,0:140];

????? 0 ??? 1 ??? 50 ???, ??? C ???
????? USE ONLY POSITIVE INTEGERS (NOT ZERO) FOR ARRAY BOUNDS

????? 0 ??? 1 ??? 50 ???, ??? 0 ??? 1 ???
????? USE ONLY POSITIVE INTEGERS (NOT ZERO) FOR ARRAY BOUNDS

BOOLEAN ARRAY PR, R, BB[0:150], C, RR[0:150,0:150];

????? 0 ??? 1 ??? 40 ???, ??? 0 ??? 1 ???
????? USE ONLY POSITIVE INTEGERS (NOT ZERO) FOR ARRAY BOUNDS

BOOLEAN ARRAY MM[0:140,0:140]; LABEL SAMEJ ARRAY HASHM[0:140];

????? 0 ??? 1 ??? 40 ???, ??? FORMAT ???
????? USE ONLY POSITIVE INTEGERS (NOT ZERO) FOR ARRAY BOUNDS

FORMAT F1 ( 40(X1,I2)/);
DEFINE TIL = +1 STEP 1 UNTIL $; FORI= FOR I TIL N DO$,
FORJ = FOR J TIL N DO$,
FORM = FOR M TIL N DO$;

????? 0 ??? 1 ??? 40 ???, ??? FORMAT ???
????? USE ONLY POSITIVE INTEGERS (NOT ZERO) FOR ARRAY BOUNDS

PROCEDURE WRITMATRIX EM, N,.Title) VALUE N, Title;
INTEGER NJ ALPHA T1, TLE; BOOLEAN ARRAY M[0,0];

????? 0 ??? 0 ??? 1 ???
????? USE ONLY POSITIVE INTEGERS (NOT ZERO) FOR ARRAY BOUNDS

BEGIN INTEGER I,J;
WRITE('<2A6," MATRIX"//40(X1,I2)>> T1,TLE, FORI I);
WRITE('<'/>>);
FORI WRITE('<40(X1,I2)>> FORJ M[I,J]) ;
WRITE('<'/>>));
END WRITMATRIX;

PROCEDURE SPANTREE(ROOT,N,C,TREE); VALUE ROOT,N;
INTEGER ROOT, N; BOOLEAN ARRAY CTREE, C[0..0];

*** BEGIN ***

USE ONLY POSITIVE INTEGERS (NOT ZERO) FOR ARRAY BOUNDS

BEGIN

BOOLEAN ARRAY PB[0..0], LEV[0..0];

FORJ PB[J] = TRUE;

LEV[0..0] = TRUE;


I = 0;

NXLEV:

FORJ IF LEV[I..J] THEN


END;

FORK IF NOT( BB[K] EQV PAR[K] ) THEN

BEGIN FORJ PB[J] = BB[J]; I = I + 1; GO TO NXLEV END;

END SPANTREE;

PROCEDURE SEPEdge (C, N, II, JJ, SNUM);

COMMENT THIS ROUTINE USES TREE INTERSECTION ALGORITHM OF I. PCOL

TO FIND SEPARATING EDGES OF GRAPH C. II[ ] AND JJ[ ] GIVE THE

ENDPOINTS OF THE BRANCHES AND SNUM THEIR NUMBER;

VALUE N; INTEGER N, SNUM; INTEGER ARRAY II[0..0]

*** BEGIN ***

BOOLEAN ARRAY C[0..0];

*** BEGIN ***

USE ONLY POSITIVE INTEGERS (NOT ZERO) FOR ARRAY BOUNDS

BEGIN

BOOLEAN ARRAY CTREE1, CTREE2 [0..0];

SNUM = 0;

SPANTREE (I, N, C, CTREE1);

SNUM = 0;

FORI FORJ SNUM = SNUM + (IF CTREE1[I..J] THEN 1 ELSE 0);

SPANTREE (N, N, C, CTREE2);

- 42 -
K+OJ
FORI FOR J TIL I DO IF (CTREE1[I,J] OR CTREE1[J,I]) THEN
  IF (CTREE2[I,J] OR CTREE2[J,I]) THEN
    BEGIN K+K+1; I1[I]+I; J1[I]+J END;
    WHILE K>0 DO
      BEGIN C[I1[K],J1[K]]=FALSE;
      WRITE(<"ITERATION IN WHILE LOOP">)
      SPANTREE(I, I, C, CTREE1);
      SUM=0; FOR FORJ SUM=SUM+IF CTREE1[I,J] THEN 1 ELSE 0;
    C[I1[K],J1[K]]=TRUE;
    IF SUM SNUM THEN
      BEGIN SNUM=SNUM+1; I1[I]+I1[K]+1; J1[I]+J1[K];
      K+K+1;
      END ELSE
      BEGIN K1+K; K+O;
      FOR T+1 STEP 1 UNTIL K1 DO
      IF CTREE1[I1[T],J1[T]] OR CTREE1[J1[T], I1[T]] THEN
        BEGIN K+K+1; I1[K]+I1[T]; J1[K]+J1[T]; END;
      END;
    END;
  END;
END SEPEDGE;

### FORTRAN DEMANDS A FORMAT. WRITE ONE PLEASE

READ (N, FORI FORJ C[I,J]);
WRIT_MATRIX(C,N,"CONNECTIVITY");
SPANTREE(SR, N, C, RR); WRIT_MATRIX(RR,N,"SPANNING");
SFEDGE (C, N, II, JJ, SNUM);
WRITE(<"SEPARATING EDGES"/>); WRITE(<214>, FOR I TIL SNUM
DC [I[I], JJ[I]]);
END.

INPUT PHASE TOOK 8.82 SECONDS.
C FOR SOME STRANGE REASON THE SCANNER CHOKES ON THE FIRST 5 BEGIN

30 FORMAT (40(1x,i2))

* SUBROUTINE WRTMA(N,VN,VTI,VTLE)
* REAL VTI,VTLE
INTEGER VN
COMMON /CMO302/ TITLE,N,M(999,999)
REAL TITLE
INTEGER N
LOGICAL M

N=VN
TI=VTI
TLE=VTLE
COMMON /CMO302/ TITLE,N,M(999,999)
REAL TITLE
INTEGER N
LOGICAL M
INTEGER I,J

40 FORMAT (2a6, 7H MATRIX//40(1x,i2))
WRITE (6, 40) TITLE(I, I=1,N,1)

50 FORMAT (///)
WRITE (6, 50)
DO 60 I=1,N,1

60 FORMAT (40(1x,i2))
WRITE (6, 70) (M(I,J), J=1,N,1)
CONTINUE

80 FORMAT (///)
WRITE (6, 80)
WRITEMATRIX
RETURN

END

SUBROUTINE SPANTR(VROOT,VN,C,CTREE)
* INTEGER VROOT,VN
COMMON /CMO504/ N,ROOT,CTREE(999,999),C(999,999)
INTEGER N,ROOT
LOGICAL CTREE,C

ROOT=VROOT
N=VN
COMMON /CMO504/ N,ROOT,CTREE(999,999),C(999,999)
INTEGER N,ROOT
LOGICAL CTREE,C
INTEGER I,J,K
LOGICAL PBC(N),BB(N),LEV(N,N)
DO 100 K=1,N,1
    BR(K)='.FALSE.'
    PBC(K)=BB(K)
    CONTINUE
    LEV(0(ROOT))='.TRUE.'
100
BB(ROOT)=.FALSE.
PBBC(ROOT)=BB(ROOT)
DO 110 J=1,N+1
    DO 120 K=1,N+1
        CTREE(J,K)=.FALSE.
        CONTINUE
    120 CONTINUE
110 CONTINUE
I=0
C NXLEV
90 DO 130 J=1,N+1
    IF (.NOT.(LEV(I,J))) GO TO 140
    DO 150 K=1,N+1
        IF (.NOT.(C(J,K).AND.BB(K))) GO TO 160
        LEV(I,K)=.TRUE.
        CTREE(J,K)=.TRUE.
        BB(K)=.FALSE.
        CONTINUE
    150 CONTINUE
140 CONTINUE
130 CONTINUE
DO 170 K=1,N+1
    IF (.NOT.(.NOT.(.NOT.(BB(K) EQV BB(K))))) GO TO 180
180 CONTINUE
170 CONTINUE
C SPANTREE
RETURN
END
C THIS ROUTINE USES TREE INTERSECTION ALGORITHM OF I. PDHL
C TO FIND SEPARATING EDGES OF GRAPH C. II(J) AND JJ(J) GIVE THE
C ENDPOINTS OF THE BRANCHES AND SNUM THEIR NUMBER
* INTEGER VN
COMMON /CM0706/ SNUM,N,II(JJ(999)),JJ(999),C(999,999)
INTEGER SNUM,N,II,JJ
LOGICAL C
N=VN
COMMON /CM0706/ SNUM,N,II(JJ(999)),JJ(999),C(999,999)
INTEGER SNUM,N,II,JJ
LOGICAL C
INTEGER I,J,K,NUM,SUM,T,K1,J1(N),I1(N)
LOGICAL CTREE2(N,N),CTREE1(N,N)
SNUM=0
CALL SPANTREE1(N,C,CTREE1)
NUM=0
00005600
00005700
00005800
00005900
00006000
00006100
00006200
00006300
00006400
00006500
00006600
00006700
00006800
00006900
00007000
00007100
00007200
00007300
00007400
00007500
00007600
00007700
00007800
00007900
00008000
00008100
00008200
00008300
00008400
00008500
00008600
00008700
00008800
00008900
00009000
00009100
00009200
00009300
00009400
00009500
00009600
00009700
00009800
00009900
00010000
00010100
00010200
00010300
00010400
00010500
00010600
00010700
00010800
00010900
00011000
DO 200 I=1,N+1
   DO 210 J=1,N+1
      NUM=NUM+(CTREE1(I,J))
   CONTINUE
210 CONTINUE
200 CONTINUE
CALL SPANTR(N,N,C,CTREE1)
K=0
DO 220 I=1,N+1
   DO 230 J=1,N+1
      IF (.NOT.((.NOT.(CTREE1(I,J)),OR,CTREE1(J,I)))) GO TO 240
      IF (.NOT.((.NOT.(CTREE2(I,J)),OR,CTREE2(J,I)))) GO TO 250
      K=K+1
      II(K)=I
      JJ(K)=J
   CONTINUE
230 CONTINUE
220 CONTINUE
250 IF (.NOT.(K.EQ.0)) GO TO 270
   C(I1(K),J1(K))=.FALSE.
270 FORMAT (23HITERATION IN WHILE LOOP)
WRITE (6, 280)
CALL SPANTR(1,N,C,CTREE1)
SUM=0
DO 290 I=1,N+1
   DO 300 J=1,N+1
      SUM=SUM+(CTREE1(I,J))
   CONTINUE
300 CONTINUE
290 CONTINUE
   C(I1(K),J1(K))=.TRUE.
   IF (.NOT.(SUM.NE.NUM)) GO TO 310
   SNUM=SNUM+1
   II(SNUM)=II(K)
   JJ(SNUM)=JJ(K)
   K=K+1
310 GO TO 320
K1=K
K=0
DO 330 T=1,K+1
   IF (.NOT.((.NOT.(CTREE1(I1(T),J1(T)),OR,CTREE1(J1(T),I1(T))))) GO TO 340
   K=K+1
   II(K)=I1(T)
   JJ(K)=J1(T)
340 CONTINUE
330 CONTINUE
320 CONTINUE
GO TO 260
270 CONTINUE
C
SEPEEDGE
RETURN
END
RETURN
END

REAL HASHM(40)
INTEGER I,J,K,M,N,NOC, JJ(40), II(40), SNUM
LOGICAL COMP(20,20), SN(40,40), SXM(40,40), BB(50), R(50), PR(50), RR(50)
C(50,50), MM(40,40)

READ (5,00000) N, (C(I,J), J=1,N), I=1,N)
CALL WRITMACC(N, 6HCONNEC, 6HTIVITY)
CALL SPANTRC5(N, C, RR)
CALL WRITMACRR(N, 6HSFANN, 6HNG)
CALL SEPEDGCC(N,II, JJ, SNUM)

350 FORMAT (16HSEPAREATING EDGES//)
WRITE (6, 350)
360 FORMAT (214)
WRITE (6, 360) (II(I), JJ(I), I=1,SNUM+1)
RETURN
END
TIME IS 3:57:53  6/2/67

77 INPUT CARDS WERE TRANSLATED, PRODUCING 185 OUTPUT CARDS, AN INCREASE OF 140%  
23 OF THE OUTPUT CARDS WERE FLAGGED, BATTING AVERAGE: 0.876

ELAPSED TRANSLATION TIME = 11.08 SECONDS.

INFO ABSORBED 63 IDENTIFIERS, WHICH USED 534 WORDS; THEY WERE USED 183 TIMES IN 46 STATEMENTS

1271 SCANS; ITEMS/STATEMENT: 28

13 VARIABLES (20%) WERE PUT INTO 3 COMMON BLOCKS (4/BLOCK)

RETRIEVALS--- IDENTIFIERS 787  BLOCKS 7  COMMON 9
FOR SOME STRANGE REASON THE SCANNER CHOKES ON THE FIRST "BEGIN" 
BEGIN FILE IN TAPE "ORDERING" (2, 56, 10); FILE OUT P 0 (2, 10); 
FORMAT OUT C ( 3(A6, A4""=W99X][ " I4 "#, " ) "000"311 "00")

***** 0   *** 10  *** ]   *** ,  *** B   ***
***** USE ONLY POSITIVE INTEGERS (NOT ZERO) FOR ARRAY BOUNDS

***** 0   *** 80  *** ]   *** ,  *** N   ***
***** USE ONLY POSITIVE INTEGERS (NOT ZERO) FOR ARRAY BOUNDS

***** 0   *** 5   *** ]   *** ,  *** INTEG ***
***** USE ONLY POSITIVE INTEGERS (NOT ZERO) FOR ARRAY BOUNDS

ALPHA ARRAY A [0:10],  B [0:80],  N [0:15]; INTEGER I,J,K,X,S

STREAM *** PROCED *** SPREAD *** (   *** X   *** ,  *** 
STREAM PROCEDURES ARE TOO ADVANCED FOR FORTRAN

STREAM PROCEDURE SPREAD (X,B); BEGIN SI+X; DI +B; 2 ( 40 (DI + DI+7; DS + CHR)) END;
INTEGER S100, S10;
S + O; S10 + 2; S100 + 7;
FOR K=0 STEP 1 UNTIL 7300

***** ,  *** A   [   *** *   *** ]   *** )   ***
***** FORTRAN DEMANDS A FORMAT. WRITE ONE PLEASE

READ (TAPE, 10, A[*]);

***** *  *** ]   *** )   *** ;   *** SPREAD *** (   ***
----- PRIMARY HAS BEEN FED A SPECIAL CHARACTER

***** *  *** ]   *** ,  *** B   *** [   *** *   ***
----- PRIMARY HAS BEEN FED A SPECIAL CHARACTER

SPREAD (A[*], B[*]);

***** *  *** ]   *** )   *** ;   *** FOR   *** X   ***
----- PRIMARY HAS BEEN FED A SPECIAL CHARACTER

FOR X=0 STEP 3 UNTIL 800 DO BEGIN 
FOR K=0 STEP 1 UNTIL 5 DO N(K) + " ";

FOR I=0 STEP 2,4 DO BEGIN 
WHILE B[I] # "" DO BEGIN 
CASE K OF BEGIN 
N[ I ],[1216] > B[I];

***** ,  *** [   *** 18   *** :   *** 6   *** ]
****** ALTRAN EXPECTED "*" AFTER PRIMARY [STMT]
N[I],[1816] + B[J];
****** ALTRAN EXPECTED "*" AFTER PRIMARY [STMT]
N[I],[2416] + B[J];
****** ALTRAN EXPECTED "*" AFTER PRIMARY [STMT]
N[I],[3016] + B[J];
****** ALTRAN EXPECTED "*" AFTER PRIMARY [STMT]
N[I],[3616] + B[J];
****** ALTRAN EXPECTED "*" AFTER PRIMARY [STMT]
N[I],[4216] + B[J];
****** ALTRAN EXPECTED "*" AFTER PRIMARY [STMT]
N[I+1],[2416] + B[J];
****** ALTRAN EXPECTED "*" AFTER PRIMARY [STMT]
N[I+1],[3016] + B[J];
****** ALTRAN EXPECTED "*" AFTER PRIMARY [STMT]
N[I+1],[3616] + B[J];
****** ALTRAN EXPECTED "*" AFTER PRIMARY [STMT]
N[I+1],[4216] + B[J];
END CASE;
K = K+1;
IF J + J+1 ≥ 72 THEN BEGIN
****** FORTRAN DEMANDS A FORMAT. WRITE ONE PLEASE
READ (TAPE, 10, A[*]);
****** PRIMARY HAS BEEN FED A SPECIAL CHARACTER
INPUT PHASE TOOK 4.55 SECONDS.
C FOR TAPE 7HOCRDMING(2,56;10PO(2;10ER CHOKES ON THE FIRST 5HBEGIN
  10 FORMAT (3(A6;A4 6H=199X(14 4H);, ) 3H000311 2H00)
  SUBROUTINE SPREAD(X,B)
     COMMON /CM0201/ B(80),X
     REAL B
     INTEGER X
     SI=X
     DI=R
     2 **** ( **** 40 **** ( **** DI=DI+7
     DS=CHR **** ) **** **** ) ****
     RETURN
     COMMON /CM0201/ B(80),X
     REAL B
     INTEGER X
     REAL A(10),N(5)
     INTEGER I,J,K,S,S100,S10
     S=0
     S10=2
     S100=7
     DO 20 K=0,73,1
  10      READ (5,00000) A(*)
     20 CONTINUE
      CALL SPREAD(A(*),B(*))
      DO 30 X=0,800,3
      DO 40 K=0,5,1
          N(K)= 6H
          CONTINUE
          K=1
      60 DO 50 I=0,2,4
          IF (.NOT.((B(J);NE.; 1H;)) GO TO 70
          CASEKOFBEGINN[I]))) ]I16] B[ ]) 2416] B(J)
          N[I;](3616] B(J)
          N[I;](4216] B(J)
          N[I+1;](2416] B(J)
          N[I+1;](3016] B(J)
          N[I+1;](3616] B(J)
          N[I+1;](4216] B(J)
      CASE
      GO TO 60
      70 K=K+1
      J=J+1
      IF (.NOT.(J;GE.;72)) GO TO 80
      CASE
      GO TO 60
      80 CASE
      GO TO 60
      50 CONTINUE
      10 CASE
      GO TO 60
      50 CONTINUE
J=J+1
K=1
C
THREE
30 CONTINUE
S=S+2
IF (.NOT. (S.EQ.10)) GO TO 90
S10=S10+1
IF (.NOT. (S10.EQ.10)) GO TO 100
S100=S100+1
S10=0
100 S=0
C
N(0),N(1),X,N(2),N(3),X+1,N(4),N(5),X+2,S100,S10,S
C
90 WRITE (6, 10) N(0),N(1),X,N(2),N(3),X+1,N(4),N(5),X+2,S100,S10,S
C
RETURN
END
TIME IS 3:56:36  6/ 2/67

41 INPUT CARDS WERE TRANSLATED, PRODUCING 71 OUTPUT CARDS, AN INCREASE OF 73%
26 OF THE OUTPUT CARDS WERE FLAGGED, BATTING AVERAGE 0.634

ELAPSED TRANSLATION TIME = 5.65 SECONDS.

INFO ABSORBED 14 IDENTIFIERS, WHICH USED 111 WORDS. THEY WERE USED 57 TIMES IN 30 STMTS;

596 SCANS; ITEMS/STMT: 20
2 VARIABLES (14%) WERE PUT INTO 1 COMMON BLOCKS (2/LOCK)

RETRIEVALS--- IDENTIFIERS 20A  BLOCKS 2  COMMON 3
Fri. May 19

THIS PROGRAM TRANSLATES A LOT OF ALGOL INTO FORTRAN

BEGIN
  ALPHA G00001;
  ALPHA NEXT1; COMMENT NEXT NON-BLANK CHARACTER TO BE SCANNED
  ALPHA SEQ; COMMENT SEQUENCE NUMBER;
  ALPHA REL;
  ALPHA HERR; H1, H2, H3, H4, H5, H6, H7, H8, H9, H10, H11, H12;
  ALPHA HERE; c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12;
  ALPHA ARRAY INFO [0115], 012551;
  ALPHA ARRAY DIRECTIVE [0115];
  ALPHA ARRAY FRESH, PIFFLE [0115];
  DEFINE NTABLE = 5, NLONG = 5, LONGBZ = 91;
  ALPHA ARRAY TABLE [01NTABLE], LONGUES [01NLONG, 01LONGBZ];
  ALPHA ARRAY OUTSAVE [0115, 0118]; INTEGER 0J;
  BOOLEAN PUNCHING, LISTING, SOURCE, OLDIF;
  BOOLEAN DEFINING, ALLIN, BROKEN, SPECIFICATIONS, CC, 029;
  BOOLEAN EXT, UNIT, EAMSTRAY, WRITEDISK, BOOLE, NONE;
  BOOLEAN SUPPRESSDIMENSIONINFORMATION, GROUP;
  DEFINE PSYM = 0#, PNUM = 1#, SYM = 2#, NM8 = 3#, SCHR = 4#;
  DEFINE STR = 5#, PSTR = 6#, FSTR = 7#;
  DEFINE INFORMLNGTH = 256 #;
  DEFINE INFOBUFFER = 3840 #;
  DEFINE REALT = 1#, INTEGRAT = 2#, BOOLEANT = 3#, OTHER = 4#;
  DEFINE LABELT = 5#, FILET = 6#, DEFINES = 7#;
  DEFINE DEFINET = 7#;
  DEFINE IDS = 1#, ARRAYS = 2#, LABELS = 3#, PROS = 4#;
  DEFINE OWN = 5#, SWITCHS = 6#, DEFINES = 7#;
  DEFINE FORMATS = 8#, FILES = 9#, LISTS = 10#;
  DEFINE OLINK = 4#;
  DEFINE LINKB = [3612], RB = [3614], NB = [4018];
  DEFINE COMM = [1212];
  DEFINE INFOIN = INFO, NEXTINFO, RB, NEXTINFO, NB #;
  DEFINE INFOBLOCKINDEX = INFO, BLOCKINDEX, RB, BLOCKINDEX, NB #;
  DEFINE INFOCOORDINDEX = INFO, COMINDEX, RB, COMINDEX, NB #;
  DEFINE INFOBUF = INFO, 1510 #;
  DEFINE BLOCKITEM = BC #;
  DEFINE BLOCKINDEX = BLOCKSTACK, BLOCKCOUNT #;
  DEFINE IS = 1, ISN'T = 2 #;
  DEFINE LESS THAN = 1<, STER = 1%, NOTLESS = 1# #,
  DEFINE EQUAL = 1=, NDIQUEAL = 1# #,
  DEFINE GTRTHAN = 1>, GTEQ = 1# #,
  DEFINE SEMICOLON = 1; #,
  DEFINE PAREN = 1(, SPAREN = 1) #,
  DEFINE BRACKET = 1[, SBRACKET = 1] #,
  DEFINE COLON = 1:, BLANK = 1 #,
  DEFINE PLUS = 1+, MINUS = 1- #,
  DEFINE ARROW = 1| #,
  DEFINE SLASH = 1/ #,
  DEFINE TIMES = 1* #,
  DEFINE DOLLAR = 1$ #,
  DEFINE

- 55 -
ELSE IF HERE IS "47090 " THEN CC = TRUE
ELSE IF HERE IS "3360 " THEN D29 = TRUE
END UNTIL HERE IS PERCENT;
IF NOT (PUNCHING OR LISTING) THEN LISTING = TRUE;
END CONTROL:
%---------------------------------------------
% INPUT PROCEDURES
%---------------------------------------------
STREAM PROCEDURE SCN (X,S,VOID, TAX, THIN, THISTAX, ESCAPE, CLASS, BUMP); VALUE THISTAX, ESCAPE, CLASS; BUMP;
BEGIN
LABEL SIXTY, DIGITS, SYMBOL, ROUNCE, START, STRING,
PASSPERCENT, BREAKS, BREAKSYM, BREAKNUM, BREAKSTR;
SI + ESCAPE; ESCAPE + TALLY; CI + CI + THISTAX;
GO TO BREAKS; GO TO BREAKS; GO TO START;
GO TO START; GO TO START; GO TO BREAKS;
SI + SI + 1; SI + SI + BUMP; %PASS PERCENT AND PREVIOUS
DI + X; DI + DI + SI; DI + DI + BUMP;
TALLY + BUMP; GO TO PASSPERCENT;
BREAKS:
TALLY + CLASS; COMMENT SET TALLY TO PREVIOUS LENGTH;
SI + SI + CLASS; BUMP+SI; SI + LOC BUMP; DI+S;
DI + X; DI + DI + 2; DI + DI + CLASS;
COMMENT BACKSPACE SI SO THAT NEXT SCANS UPDATE WORKS;
SI + BUMP; SI + SI + CLASS;
CI + CI + THISTAX; GO TO BREAKSYM; GO TO BREAKNUM;
TALLY+O; TALLY+O; TALLY+O; TALLY+O;
BREAKSTR: IF SC="" THEN GO TO STRING;
BUMP + TALLY; TALLY + STR; GO TO BOUNCE;
BREAKSYM: IF SC=ALPHA THEN GO TO SYMBOL;
BUMP + TALLY; TALLY + SYM; GO TO ROUNCE;
BREAKNUM: IF SC=0 THEN GO TO DIGITS;
BUMP + TALLY; TALLY + NUM; GO TO ROUNCE;
START:
SI + SI + BUMP; % UPDATE SI
BUMP + SI; SI + LOC BUMP; DI + SI; DS + WD;
S (SC IN THE REAL WORLD) NOW KNOWS WHERE THE SCANNER IS
SI + BUMP;
IF SC="" THEN BEGIN
SIXTY: SI + SI + 1; TALLY + TALLY + 1; IF SC = " " THEN00020000
GO TO SIXTY END;
ESCAPE + TALLY; TALLY + O; % THE NEW WASTE;
DI + X; DI + DI + 2; % (THE RESULT GOES INTO X)
IF SC = "0" THEN GO TO DIGITS;
IF SC = ALPHA THEN GO TO SYMBOL;
STRING: IF SC=""THEN BEGIN BUMP+TALLY;TALLY+PSTR;GO BOUNCE END;
PASSPERCENT: DS + CHRI; TALLY + TALLY + 1;
IF SC = "1" THEN GO TO STRING;
BUMP + TALLY;
TALLY + STR; SI + SI + 1; % FOR "NEXT" PAST "
GO TO BOUNCE END SCANNING STRINGS;
DS + CHRI; TALLY + 1; BUMP + TALLY;
TALLY + CHRI; GO TO ROUNCE;
DIGITS:
DS + CHRI; TALLY + TALLY + 1; IF SC = "0" THEN GO DIGITS;
00021500
0217
PROCEDURE COMMODE; FORWARD
 INTEGER STREAM
 BEGIN
 SI = LOC ESCAPE; DI + VOID; DS + WDS;
 % (THE OUTSIDE WORLD GETS THE NEW WASTE VALUE THROUGH VOID)
 DI + TAX; DS + WDS; % (OUT GOES THE CLASS OF HERE)
 DI + THINJ DS + WDS; % (OUT GOES THE THICKNESS OF HERE)
 DS + 2 CHRj % (EACH ITEM IS PREFIXED BY ITS LENGTH)
 END SCN THE ALMIGHTY;
 INTEGER STREAM PROCEDURE OKCOMMENT (S, D, R, SV, STOP); VALUE R, SV;
 BEGIN LABEL WHOA, FULL;
 SI + SVI DI + DJ DI+DI+2;
 R (IF SC = "j") THEN JUMP OUT TO WHOA;
 END PROEDURE OKCOMMENT;
 PROCEDURE CHANGE CARDSj BEGIN
 IF SOURCE THEN READING (INFIL(O), OUTFIL(O), OUTFIL);
 RELEASE (INFIL); INPUT + INPUT + 1;
 ZOT ("M", INFIL (O)) SC + MKABS (INFIL(O));
 L + WASTE + DI;
 COLUMN + 1 END CHANGE CARDS;
 INTEGER IOTA, CARDS;
 PROCEDURE STEPIT (THIS, HERE); INTEGER THIS; ALPHA HERE; BEGIN
 LABEL SCNN;
 IF LON PREVIOUS CARD = 0 THEN HERE + "j"
 IF L + L + WASTE > 63 THEN BEGIN
 SC + SC+8j L + L = 64 END EXTRA LONG STRINGS;
 SCNN: SCN (HERE, SC, WASTE, THIS, L, THIS, SC,
 LON PREVIOUS CARD, L); NU + L;
 COLUMN + COLUMN + L + WASTE = LON PREVIOUS CARDj
 IF THIS IS PSTR THEN IF NOT BROKEN + COLUMN272 THEN BEGIN
 THIS + FPSTj GO TO SCN ENDj
 IF L = 0 THEN L + 64j COMMENT IN CASE OF EXTRA LONG STRINGS
 IF THIS IS PSYM OR THIS IS PNUM OR THIS IS PSTR THEN BEGIN00026400
 LON PREVIOUS CARD + L + LON PREVIOUS CARD;
 CHANGE CARDSj
 CARDS+CARDS+1; STEPIT (THIS, HERE); CARDS+CARDS+1;
 LON PREVIOUS CARD + 0 END SYMBOL SPLIT ACROSS 2 CROSS100026800
 IF HERE THEN COMMENT THEN BEGIN COMMODE;
 HERE + "j" GO TO SCN END CM100026890

 - 59 -
IF HERE IS DOLLAR THEN CONTROLCARD (THIS, HFRE) ELSE
  00026900 0273
IF HFRE IS PERCENT THEN IF CARDSSD THEN CHANGECARDS
  00027000 0274
IF THIS IS STR THEN IF CARDSSD THEN BEGIN
  00027100 0275
  WASTE + WASTE + (IF NOT BROKEN THEN 2 ELSE 1);+ HERE* (616) + NU + NU+3; IF L = 10 THEN HERE.[1216] + L DIV 10;
  00027200 0276
HERE.[18+6] + ENTER (L MOD 10 + ,1); COLUMN + COLUMN + (IF NOT BROKEN THEN 2 ELSE 1); END
  00027300 0277
BROKEN = FALSE;
  00027400 0278
END STRING SCANNER;
  00027500 0279
IOTA = IOTA+1; COMMENT COUNT NUMBER OF SCANS MADE;
  00027600 0280
END SCAN THE COMPLETE INPUT PROCESSOR;
  00027700 0281
PROCEDURE SCAN: BEGIN
  00027800 0282
  DEFINE THSS = TTABLE [NTAXBLE] #;
  00027900 0283
  LABEL TWICE;
  00027990 0284
  MOVE (NTABLE, T able(1), T a ble(0));
  00028000 0285
  MOVE (NTABLE, T able(1), T a ble(0));
  00028100 0286
  IF HERE + TABLE [0] < 0 THEN
  00028200 0287
  MOVE (LON GSIZE+1, LON GJES [*HERE,*], HERE);
  00028300 0288
  IF NOT ALLIN THEN BEGIN
  00028400 0289
  STEP IT (THSS, HERR);
  00028500 0290
  IF THSS IS SYM THEN BEGIN
  00028600 0291
  INTEGER SVIJ;
  00028700 0292
  SVJ * JJ GET (HERR);
  00028800 0293
  IF IDTYPE IS DEFINET THEN BEGIN
  00028900 0294
  DEFINING * TRUE;
  00029000 0295
  SAVEC [IX] + SC; SAVEC [DX+1] + LJ
  00029100 0296
  SAVEC [DX+2] + WASTE + DIX + DIX+3;
  00029200 0297
  SC + MKABS (IDSTRING) L + WASTE + OJ
  00029300 0298
  I + SVIJ
  00029400 0299
  GO TO TWICE END DEFINED SYMBOL;
  00029500 0300
  I * SVJ END SYMBOL;
  00029600 0301
  IF THSS IS SCHR THEN BEGIN
  00029700 0302
  IF HERR IS PERCENT THEN GO TO TWICE;
  00029800 0303
  IF HERR IS CROSSHATCH AND DEFINING THEN BEGIN
  00029900 0304
  SC + SAVEC [DX+DIX+3];
  00030000 0305
  L + SAVEC [DX+1]; WASTE + SAVEC [DX+2];
  00030100 0306
  DEFINING * DIX+O;
  00030200 0307
  GO TO TWICE END DEFINED STRING;
  00030300 0308
  IF HERR IS PERIOD THEN
  00030400 0309
  IF ALLIN * TABLE [NTAXBLE-1] = ENDW THEN BEGIN
  00030500 0310
  IF SOURCE THEN READBACK (INFILE);-
  00030600 0311
  OUTFILE(0), OUTFILE);
  00030700 0312
  CLOSE (INFILE, RE FILE); END VERY LAST CARD; 00030800 0313
  00030900 0314
  END SPECIAL CHARACTERS JUST COMING IN;
  00031000 0315
IF NU*6 THEN TABLE [NTAXBLE] * HERR
  00031100 0316
ELSE BEGIN TABLE [NTAXBLE] * -(IF GJ + GJ+1 = NLONG
  00031200 0317
  THEN GJ ELSE GJ+1));
  00031300 0318
MOVE (LON GSIZE+1, HERR, LON GJES [GJ,*], FND SYM 6j;
  00031400 0319
END NOT ALL ITEMS HAVE BEEN BROUGHT IN BY THE SCANNER;
  00031500 0320
  00031600 0321
  00031700 0322
IF NFXT + TABLE[11] < 0 THEN NEXT + LON GJES [*NFXT,0];
  00031800 0323
NU + HERE* (616); THIS + TABLE [0];
  00031900 0324
END BOTTOM SIDE SCANNER;
  00032000 0325
  00032100 0326
  00032200 0327
INTEGER PROCEDURE SCRAMBLE (II); VALUE II; INTEGER II;
  00032300 0328
SCRAMBLE + (I*543876,024916),(2618);
  00032400 0329
- 60 -
ALPHA STREAM PROCEDURE UP (Q, DQ); VALUE DQ; BEGIN
SI + LOC DQ; DI + Q; DS = Q ADD;
DI + LOC UP; SI + Q; DS + WDS END;

BOOLEAN STREAM PROCEDURE EJECTACARD (OUTPUT, MARG); VALUE MARG; BEGIN
LABEL WRAPUP, LABELLED, FIN;
SI + OUTPUT;
SI IF SC = "Y" THEN JUMP OUT TO LABELLED; SI + SI + 1;
I ( IF SC = "Y" THEN JUMP OUT 2 TO WRAPUP; SI + SI + 1 );
DI + OUTPUT; DI + DI + 1; MARG ( DI + DI + 1 );
DS + 8 LIT "CONTINUE";
WRAPUP;
TALLY + 1; EJECTACARD + TALLY;
FINI END EJECTACARD;

STREAM PROCEDURE D25CHARACTERSET (P); BEGIN
LOCAL D9, D8, D7, D6, D5, D4, D3, D2, D1; LABEL SKPP, OK;
SI = P; DI = LOC D1;
2 ( 36 ( IF SC = "=" THEN BEGIN
DS + LIT "X" J GO TO SKPP END;
IF SC = "=" THEN BEGIN
DS + LIT "X" J GO TO SKPP END;
IF SC = "(" THEN BEGIN
DS + LIT "[" J GO TO SKPP END;
IF SC = "") THEN BEGIN
DS + LIT ""]" J GO TO SKPP END;
DS + LIT "[" J GO TO OK END;
DO IF WRITEDISK THEN BEGIN
IF EJECTACARD (PIFFLE [1*], MARGIN=SKIP) THEN BEGIN
PIFFLE [10] = RFAL (LISTING);
PIFFLE [11] = RFAL (PUNCHING);
WRITE (PARTIAL, 15, PIFFLE [*]);
MOVE (15, FRESH[*], PIFFLE[*]);
END;
PROCEDURE ENEW; BEGIN
00035600 0360
END;
ELSE BEGIN
TOTAL + TOTAL + 1;
IF PIFFLE [9] # 0 THEN ZOT (PIFFLE [9], PIFFLE [*]);
PIFFLE [9] = UP (SEQ, "[100]");
IF D9 THEN D25CHARACTERSET (PIFFLE[*]);
IF BOOLEAN (PIFFLE [11]) THEN BEGIN
MOVE (10, PIFFLE [1*], P(0)); RELEASE (P) END;
IF BOOLEAN (PIFFLE [10]) THEN BEGIN
MOVE (10, PIFFLE [1*], OUTFIL(0)); RELEASE (OUTFIL) END;
MOVE (10, FRESH[*], PIFFLE[*]);
END USER OUTPUT;
W + (PLACE + MARGIN), WORD; C + MARGIN, CHR
END ENEW;

PROCEDURE ENEW; BEGIN
IF WRITEDISK THEN BEGIN
IF EJECTACARD (PIFFLE [*], MARGIN=SKIP) THEN BEGIN
PIFFLE [10] = RFAL (LISTING);
PIFFLE [11] = RFAL (PUNCHING);
WRITE (PARTIAL, 15, PIFFLE [*]);
MOVE (15, FRESH[*], PIFFLE[*]);
END;
PROCEDURE ENEW; BEGIN
00035600 0360
END;
ELSE BEGIN
TOTAL + TOTAL + 1;
IF PIFFLE [9] # 0 THEN ZOT (PIFFLE [9], PIFFLE [*]);
PIFFLE [9] = UP (SEQ, "[100]");
IF D9 THEN D25CHARACTERSET (PIFFLE[*]);
IF BOOLEAN (PIFFLE [11]) THEN BEGIN
MOVE (10, PIFFLE [1*], P(0)); RELEASE (P) END;
IF BOOLEAN (PIFFLE [10]) THEN BEGIN
MOVE (10, PIFFLE [1*], OUTFIL(0)); RELEASE (OUTFIL) END;
MOVE (10, FRESH[*], PIFFLE[*]);
END USER OUTPUT;
W + (PLACE + MARGIN), WORD; C + MARGIN, CHR
END ENEW;

PROCEDURE ENEW; BEGIN
IF WRITEDISK THEN BEGIN
IF EJECTACARD (PIFFLE [*], MARGIN=SKIP) THEN BEGIN
PIFFLE [10] = RFAL (LISTING);
PIFFLE [11] = RFAL (PUNCHING);
WRITE (PARTIAL, 15, PIFFLE [*]);
MOVE (15, FRESH[*], PIFFLE[*]);
END;
PROCEDURE ENEW; BEGIN
00035600 0360
END;
ELSE BEGIN
TOTAL + TOTAL + 1;
IF PIFFLE [9] # 0 THEN ZOT (PIFFLE [9], PIFFLE [*]);
PIFFLE [9] = UP (SEQ, "[100]");
IF D9 THEN D25CHARACTERSET (PIFFLE[*]);
IF BOOLEAN (PIFFLE [11]) THEN BEGIN
MOVE (10, PIFFLE [1*], P(0)); RELEASE (P) END;
IF BOOLEAN (PIFFLE [10]) THEN BEGIN
MOVE (10, PIFFLE [1*], OUTFIL(0)); RELEASE (OUTFIL) END;
MOVE (10, FRESH[*], PIFFLE[*]);
END USER OUTPUT;
W + (PLACE + MARGIN), WORD; C + MARGIN, CHR
END ENEW;

PROCEDURE ENEW; BEGIN
IF WRITEDISK THEN BEGIN
IF EJECTACARD (PIFFLE [*], MARGIN=SKIP) THEN BEGIN
PIFFLE [10] = RFAL (LISTING);
PIFFLE [11] = RFAL (PUNCHING);
WRITE (PARTIAL, 15, PIFFLE [*]);
MOVE (15, FRESH[*], PIFFLE[*]);
END;
PROCEDURE ENEW; BEGIN
00035600 0360
END;
ELSE BEGIN
TOTAL + TOTAL + 1;
IF PIFFLE [9] # 0 THEN ZOT (PIFFLE [9], PIFFLE [*]);
PIFFLE [9] = UP (SEQ, "[100]");
IF D9 THEN D25CHARACTERSET (PIFFLE[*]);
IF BOOLEAN (PIFFLE [11]) THEN BEGIN
MOVE (10, PIFFLE [1*], P(0)); RELEASE (P) END;
IF BOOLEAN (PIFFLE [10]) THEN BEGIN
MOVE (10, PIFFLE [1*], OUTFIL(0)); RELEASE (OUTFIL) END;
MOVE (10, FRESH[*], PIFFLE[*]);
END USER OUTPUT;
W + (PLACE + MARGIN), WORD; C + MARGIN, CHR
END ENEW;

PROCEDURE ENEW; BEGIN
IF WRITEDISK THEN BEGIN
IF EJECTACARD (PIFFLE [*], MARGIN=SKIP) THEN BEGIN
PIFFLE [10] = RFAL (LISTING);
PIFFLE [11] = RFAL (PUNCHING);
WRITE (PARTIAL, 15, PIFFLE [*]);
MOVE (15, FRESH[*], PIFFLE[*]);
END;
PROCEDURE ENEW; BEGIN
00035600 0360
END;
ELSE BEGIN
TOTAL + TOTAL + 1;
IF PIFFLE [9] # 0 THEN ZOT (PIFFLE [9], PIFFLE [*]);
PIFFLE [9] = UP (SEQ, "[100]");
IF D9 THEN D25CHARACTERSET (PIFFLE[*]);
IF BOOLEAN (PIFFLE [11]) THEN BEGIN
MOVE (10, PIFFLE [1*], P(0)); RELEASE (P) END;
IF BOOLEAN (PIFFLE [10]) THEN BEGIN
MOVE (10, PIFFLE [1*], OUTFIL(0)); RELEASE (OUTFIL) END;
MOVE (10, FRESH[*], PIFFLE[*]);
END USER OUTPUT;
W + (PLACE + MARGIN), WORD; C + MARGIN, CHR
END ENEW;
STREAM PROCEDURE CONTINUE (S); BEGIN DI+SI DI+DI+5; DS + LIT "-" END; 00037900 0383
PROCEDURE SAVOUT; IF DJ < 16 THEN BEGIN 00038000 0384
MOVE (16, PIFFLE[*], OUTSAVE [OJ,*]); 00038100 0385
MOVE (10, FRESH[*], PIFFLE[*]); 00038200 0386
OUTSAVE [OJ,18] + C; 00038400 0388
OJ + DJ + 1 END
ELSE WRITE (OUTFIL, NOOJ, OJ+DJ+1); 00038600 0390
PROCEDURE RESTOROUT IF OJ < 17 THEN BEGIN 00038700 0391
MOVE (16, OUTSAVE [OJ-1,*],PIFFLE[*]); 00038800 0392
PLACE + OUTSAVE [OJ,16] + W + OUTSAVE [OJ,17]; 00038900 0393
C + OUTSAVE [OJ,18] END RESTORING THE INTERRUPTED OUTPUT 00039000 0394
ELSE OJ = OJ-1; 00039100 0395
PROCEDURE INJECT (NU,H*), VALUE NU; INTEGER NU; ALPHA HERE; BEGIN 00039200 0396
IF PLACE + NU > 72 THEN BEGIN 00039300 0397
IF NU < 65 THEN BEGIN 00039400 0398
ENEL; CONTINUE (PIFFLE[*]); 00039500 0399
END ELSE BEGIN 00039600 0400
PLACE + NU > 72 THEN BEGIN 00039700 0401
PLACE + C + 6; W + 0 END END ONE XTR00039800 0402
PLACE + NU = NPART; SEND (NPART + 71,PLACE, HERE, PIFFLE[*], C); 00039900 0403
DO BEGIN 00040000 0404
NU = NU - NPART; 00040100 0405
ENEL; CONTINUE (PIFFLE[*]); 00040200 0406
MOKSTRING ((SN + NPART+2)26A, SN.[3916], 00040300 0407
SN.[4513], HERE, 0, 0, 2, HERE, NU264+NU.[4513]); 00040400 0408
NU.[3916], NU.[4513] SEND (IF NU<65 THEN NU ELSE 65, HERE, 00040500 0409
PIFFLE[O, 6]); 00040600 0410
NPART = 65 END ITERATING LINES OF 65 EACH 00040700 0411
UNTIL NU < 65; 00040800 0412
IF NU-65 COMMENT THIS MAKES FALLTHRU CODE NIL; 00040900 0413
END CASE WHERE NU GTR THEN 65 00041000 0414
PROCEDURE EMIT (H,N); VALUE H,N; ALPHA H; INTEGER NJ BEGIN 00041100 0415
ALPHA HERE; INTEGER NU; IF C.C+NU<8 THEN BEGIN 00041200 0416
C.C+C.WORDl C.C.CHR END 00041300 0417
PROCEDURE E; BEGIN IF NU > 6 THEN IF THIS ISNT NMA THEN IF THIS 00041400 0418
ISNT STR THEN NU + 6; INJECT (NU, HERE) END F; 00041500 0419
PROCEDURE EMIT (H,N); VALUE H,J; ALPHA H; IF H.[616] < 6 THEN 00041600 0420
INJECT (H.[L616], H) ELSE INJECT (6, H); 00041700 0421
PROCEDURE EMIT (H,N); VALUE H,N; ALPHA H; INTEGER NJ BEGIN 00041800 0422
ALPHA HERE; INTEGER NU; IF H = H2 IF NU + N > 6 THEN IF THIS ISNT NMB THEN NU+6; 00041900 0423
INJECT (NU, HERE) END EMIT; 00042000 0424
PROCEDURE ECOMMENT BEGIN 00042100 0425
IF PLACE + PLACE+1 + NU > 72 THEN BEGIN 00042200 0426
ENEL; ZOT ("C", PIFFLE[*]); PLACE+1 C+1 END NXT00042300 0427
END FCOMMENT; 00042400 0428
PROCEDURE CCOMMODE BEGIN 00042500 0429
LABEL FARSIDE; INTEGER NJ; BOOLEAN STOP, START; 00042600 0430
INTEGER RANGEJ 00042700 0431
PASSCHAR (SC, WASTE+L); START + TRUE; 00042800 0432
- 62 -
SAVOUT; W+C+PLACE+O; INJECT (COLUMN=1, FRESH(0));

IF COLUMN <= 37 THEN RANGE = 37 - COLUMN;
ELSE BEGIN
  RANGE = 73 - COLUMN;
  GO TO FARSIDE END;

DO BEGIN
  IF NOT START THEN BEGIN
    W+C+PLACE+O; INJECT (COLUMN=1, FRESH(0));
    N = OKCOMMENT (SC, HERR, RANGE, SC, STOP);
    INJECT (N, HERR); RANGE = 36;
  END
  IF NOT START THEN COLUMN + 1;
  FARSIDE: ZOT ("C", PITCH[*l);
  IF NOT STOP THEN BEGIN
    OKCOMMENT (SC, HERR, RANGE, SC, STOP);
    INJECT (N, HERR);
    RANGE = 36;
  END
  IF NOT START THEN COLUMN + 37 END;
UNTIL STOP;

END COLUMN;

DO BEGIN
  IF NOT START THEN BEGIN
    W+C+PLACE+O; INJECT (COLUMN=1, FRESH(0));
    N = OKCOMMENT (SC, HERR, RANGE, SC, STOP);
    INJECT (N, HERR); RANGE = 36;
  END
  IF NOT START THEN COLUMN + 1;
  FARSIDE: ZOT ("C", PITCH[*l);
  IF NOT STOP THEN BEGIN
    OKCOMMENT (SC, HERR, RANGE, SC, STOP);
    INJECT (N, HERR);
    RANGE = 36;
  END
  IF NOT START THEN COLUMN + 37 END;
UNTIL STOP;

END COLUMN;

DO BEGIN
  IF NOT START THEN BEGIN
    W+C+PLACE+O; INJECT (COLUMN=1, FRESH(0));
    N = OKCOMMENT (SC, HERR, RANGE, SC, STOP);
    INJECT (N, HERR); RANGE = 36;
  END
  IF NOT START THEN COLUMN + 1;
  FARSIDE: ZOT ("C", PITCH[*l);
  IF NOT STOP THEN BEGIN
    OKCOMMENT (SC, HERR, RANGE, SC, STOP);
    INJECT (N, HERR);
    RANGE = 36;
  END
  IF NOT START THEN COLUMN + 37 END;
UNTIL STOP;

END COLUMN;

DO BEGIN
  IF NOT START THEN BEGIN
    W+C+PLACE+O; INJECT (COLUMN=1, FRESH(0));
    N = OKCOMMENT (SC, HERR, RANGE, SC, STOP);
    INJECT (N, HERR); RANGE = 36;
  END
  IF NOT START THEN COLUMN + 1;
  FARSIDE: ZOT ("C", PITCH[*l);
  IF NOT STOP THEN BEGIN
    OKCOMMENT (SC, HERR, RANGE, SC, STOP);
    INJECT (N, HERR);
    RANGE = 36;
  END
  IF NOT START THEN COLUMN + 37 END;
UNTIL STOP;

END COLUMN;

DO BEGIN
  IF NOT START THEN BEGIN
    W+C+PLACE+O; INJECT (COLUMN=1, FRESH(0));
    N = OKCOMMENT (SC, HERR, RANGE, SC, STOP);
    INJECT (N, HERR); RANGE = 36;
  END
  IF NOT START THEN COLUMN + 1;
  FARSIDE: ZOT ("C", PITCH[*l);
  IF NOT STOP THEN BEGIN
    OKCOMMENT (SC, HERR, RANGE, SC, STOP);
    INJECT (N, HERR);
    RANGE = 36;
  END
  IF NOT START THEN COLUMN + 37 END;
UNTIL STOP;

END COLUMN;

DO BEGIN
  IF NOT START THEN BEGIN
    W+C+PLACE+O; INJECT (COLUMN=1, FRESH(0));
    N = OKCOMMENT (SC, HERR, RANGE, SC, STOP);
    INJECT (N, HERR); RANGE = 36;
  END
  IF NOT START THEN COLUMN + 1;
  FARSIDE: ZOT ("C", PITCH[*l);
  IF NOT STOP THEN BEGIN
    OKCOMMENT (SC, HERR, RANGE, SC, STOP);
    INJECT (N, HERR);
    RANGE = 36;
  END
  IF NOT START THEN COLUMN + 37 END;
UNTIL STOP;

END COLUMN;

DO BEGIN
  IF NOT START THEN BEGIN
    W+C+PLACE+O; INJECT (COLUMN=1, FRESH(0));
    N = OKCOMMENT (SC, HERR, RANGE, SC, STOP);
    INJECT (N, HERR); RANGE = 36;
  END
  IF NOT START THEN COLUMN + 1;
  FARSIDE: ZOT ("C", PITCH[*l);
  IF NOT STOP THEN BEGIN
    OKCOMMENT (SC, HERR, RANGE, SC, STOP);
    INJECT (N, HERR);
    RANGE = 36;
  END
  IF NOT START THEN COLUMN + 37 END;
UNTIL STOP;

END COLUMN;

DO BEGIN
  IF NOT START THEN BEGIN
    W+C+PLACE+O; INJECT (COLUMN=1, FRESH(0));
    N = OKCOMMENT (SC, HERR, RANGE, SC, STOP);
    INJECT (N, HERR); RANGE = 36;
  END
  IF NOT START THEN COLUMN + 1;
  FARSIDE: ZOT ("C", PITCH[*l);
  IF NOT STOP THEN BEGIN
    OKCOMMENT (SC, HERR, RANGE, SC, STOP);
    INJECT (N, HERR);
    RANGE = 36;
  END
  IF NOT START THEN COLUMN + 37 END;
UNTIL STOP;

END COLUMN;

DO BEGIN
  IF NOT START THEN BEGIN
    W+C+PLACE+O; INJECT (COLUMN=1, FRESH(0));
    N = OKCOMMENT (SC, HERR, RANGE, SC, STOP);
    INJECT (N, HERR); RANGE = 36;
  END
  IF NOT START THEN COLUMN + 1;
  FARSIDE: ZOT ("C", PITCH[*l);
  IF NOT STOP THEN BEGIN
    OKCOMMENT (SC, HERR, RANGE, SC, STOP);
    INJECT (N, HERR);
    RANGE = 36;
  END
  IF NOT START THEN COLUMN + 37 END;
UNTIL STOP;

END COLUMN;

DO BEGIN
  IF NOT START THEN BEGIN
    W+C+PLACE+O; INJECT (COLUMN=1, FRESH(0));
    N = OKCOMMENT (SC, HERR, RANGE, SC, STOP);
    INJECT (N, HERR); RANGE = 36;
  END
  IF NOT START THEN COLUMN + 1;
  FARSIDE: ZOT ("C", PITCH[*l);
  IF NOT STOP THEN BEGIN
    OKCOMMENT (SC, HERR, RANGE, SC, STOP);
    INJECT (N, HERR);
    RANGE = 36;
  END
  IF NOT START THEN COLUMN + 37 END;
UNTIL STOP;

END COLUMN;
("**** USE ONLY POSITIVE INTEGERS (NOT ZERO) FOR ARRAY ROUND")

"S")

("**** ALTRAN EXPECTED "-----" "-----" (BE)"), )

("---- CHECK YOUR SYNTAX, DECLARATION FOUND IN THE MIDD" "00054500"

"STATEMENTS, DECKHEAD CALLED")

("**** USE EITHER AN UNSIGNED INTEGER CONSTANT OR UNSUBSCRIP" "00054600"

"TED INTEGER VARIABLE")

("**** FOR-VARIABLE MUST BE UNSUBSCRIBED INTEGER VARIABLE")

("**** THIS TRANSLATOR IS GOING TO LET YOU TRY TO TRANSLATE" "00054700"

"SWITCH-FILES, FORMATS, LISTS BY YOURSELF")

("**** NOW "-----" "-----" REALLY CONFUSED")

("**** NOT ENOUGH ERROR MESSAGES, WILLER")

PROCEDURE GOLOST (N); VALUE N;
BEGIN
INTEGER I;
WRITE (DUTFIL, WHERE, COMINGATTRACTIONS);
WRITE (OUTFIL, WHICHWAY (N)];
MARK (".");
WRITE (OUTFIL); END ERROR MESSAGE POINTER;

%------------------------------------------
% FUNDAMENTAL PROCEDURES
%------------------------------------------

PROCEDURE PUTINCOMMON; FORWARD;
PROCEDURE AE; FORWARD;
PROCEDURE ElIST; FORWARD;
PROCEDURE PRO (TYPE); INTEGER TYPE; FORWARD;
PROCEDURE EST; FORWARD;
PROCEDURE INASSIGN; FORWARD;
PROCEDURE STMT; FORWARD;
PROCEDURE COMPOUNDTAIL; FORWARD;
PROCEDURE BLOCK; FORWARD;
PROCEDURE GLOBALCHECK; IF BLOCKCOUNT > 1 THEN BEGIN
GET (HERE);
IF IDBLOCK \ BLOCKCOUNT THEN IF IDID \ 0 THEN
IF IMPROFIF \ 1 THEN IF IDTYPE \ LABEL
THEN PUTINCOMMON
ELSE GOLOST (8) END GLOBALCHECK;
BOOLEAN PROCEDURE TERMINATOR (HERE); ALPHA HERE;
IF HERE IS SEMICOLON THEN TERMINATOR \ TRUE
ELSE IF HERE IS END THEN TERMINATOR \ TRUE
ELSE IF HERE IS ELSE THEN TERMINATOR \ TRUE
ELSE IF HERE IS UNTIL THEN TERMINATOR \ TRUE;
PROCEDURE NUMBER; BEGIN
IF NMB THEN BEGIN EJ SCAN END;
IF HERE IS "1" THEN BEGIN EJ SCAN END;
IF THIS IS NMB THEN BEGIN EJ SCAN END;
IF HERE IS "1" OR HERE IS "1E" THEN BEGIN
EMIT ("1E","1") SCAN;
IF HERE IS "1" OR HERE IS "1" THEN BEGIN
EMIT ("1E","1") SCAN;
IF THIS IS NMB THEN BEGIN EJ SCAN END;
ELSE GOLOST (13) END;
END NUMBER;
PROCEDURE PRIMARY; BEGIN
LABEL SUBSCRIPTS;
IF HERE IS "1" THEN BEGIN EJ SCAN; AEJ
IF HERE \ "1" THEN BEGIN
GOLOST (10)j FLASH ("1") "") ENDJ
00055200
0558
0559
0560
0561
0562
0563
0564
0565
0566
0567
0568
0569
0570
0571
0572
0573
0574
0575
0576
0577
0578
0579
0580
0581
0582
0583
0584
0585
0586
0587
0588
0589
0590
0591
0592
0593
0594
0595
0596
0597
0598
0599
0600
0601
0602
IF NEXT ISN'T PERIOD THEN BEGIN SJ SCAN END PAST SPAREN 00059700 0603
ELSE FLASH (SBRACKET) END PARENTHESIZED PRIMARY 00059800 0604
ELSE IF HERE IS IF THEN AE 00059900 0605
ELSE IF THIS IS SYM THEN BEGIN
NUPRIM * NUPRIM + 1J 00060000 0606
IF BOOL THEN IF HERE IS "4TRUE " THEN BEGIN
HERE + "4TRUE, "; J NU + 6 END 00060200 0607
ELSE IF HERE IS "5FALSE " THEN BEGIN
EMIT (PERIOD,1) HERE="5FALSE,"; J NU+6 ENDO00060500 0610
ELSE GLOBALCHECK 00060600 0611
ELSE GLOBALCHECKJ 00060700 0612
IF NEXT IS BRACKET THEN BEGIN
SJ SCAN; HERE + PAREN; 00060800 0613
IF TABLE(2) IS COLON THEN FLASH (SBRACKET) 00060900 0614
ELSE GO TO SUBSCRIPTS END SYMROL BRACKET 00061000 0615
ELSE NEXT IS PAREN THEN BEGIN
SJ SCAN; HERE IS "6ARCTAN" THEN EMIT "4ATAN",4) 00061100 0616
ELSE HERE IS "6INT " THEN EMIT "3INT",3) 00061200 0617
ELSE HERE IS "2LN " THEN EMIT "3LOr,"3) 00061300 0618
ELSE SJ 00061400 0619
SUBSCRIPTS: SJ SCAN; ELIST;
IF HERE IS "1) " THEN HERE + "1) " 00061500 0620
ELSE IF HERE # "1) " THEN BEGIN
SJ SCAN; HERE + PAREN; 00061600 0621
END NEXT BEING PAREN WHICH MEANS FUNCTION CALL 00061700 0622
ELSE IF NEXT IS PERIOD THEN BEGIN
MARK (".*") FLASH (SBRACKET) 00061800 0623
Sj SCAN ENDSJ 00061900 0624
ELSE IF THIS IS NMB THEN NUMBER 00062000 0625
ELSE IF THIS IS STR THEN BEGIN SJ SCAN END HOLLERITH FIELD 00062100 0626
ELSE IF HERE IS "1", " OR HERE IS "10 " THEN NUMBER 00062200 0627
ELSE IF HERE IS MINUS THEN BEGIN SJ SCAN; AE END UNARY MNUS 00062300 0628
ELSE BEGIN GOTLOST (12) DO BEGIN SJ SCAN END 00062400 0629
UNTIL THIS IS SCR END NONPRIMARY TYPE ITEM 00062500 0630
PROCEDURE TERM BEGIN
LABEL NCOL,SOJ DO BEGIN
SJ SCAN; 00062600 0631
END PRIMARYJ 00062700 0632
PROCEDURE TERMJ BEGIN LABEL NCOL,S0J DO BEGIN
primaryJ SOJ 00062800 0633
IF HERE IS "1X " THEN EMIT (STAR,1) 00062900 0634
ELSE IF HERE IS SLASH THEN E 00063000 0635
END SJ SCAN; 00063100 0636
END PRIMARYJ 00063200 0637
PROCEDURE OPERATOR (HERE)J VALUE HEREJ ALPHA HEREJ BEGIN
OPERATOR + TRUEJ IF HERE ISN'T PLUS V MINUS V TIMES V 00063300 0638
OPERATOR + TRUEJ 00063400 0639
PROCEDURE OPERATOR (HERE)J VALUE HEREJ ALPHA HEREJ BEGIN
OPERATOR + TRUEJ IF HERE ISN'T PLUS V MINUS V TIMES V 00063500 0640
OPERATOR + TRUEJ 00063600 0641
PROCEDURE OPERATOR (HERE)J VALUE HEREJ ALPHA HEREJ BEGIN
OPERATOR + TRUEJ IF HERE ISN'T PLUS V MINUS V TIMES V 00063700 0642
OPERATOR + TRUEJ 00063800 0643
PROCEDURE OPERATOR (HERE)J VALUE HEREJ ALPHA HEREJ BEGIN
OPERATOR + TRUEJ IF HERE ISN'T PLUS V MINUS V TIMES V 00063900 0644
OPERATOR + TRUEJ 00064000 0645
PROCEDURE OPERATOR (HERE)J VALUE HEREJ ALPHA HEREJ BEGIN
OPERATOR + TRUEJ IF HERE ISN'T PLUS V MINUS V TIMES V 00064100 0646
OPERATOR + TRUEJ 00064200 0647
PROCEDURE OPERATOR (HERE)J VALUE HEREJ ALPHA HEREJ BEGIN
OPERATOR + TRUEJ IF HERE ISN'T PLUS V MINUS V TIMES V 00064300 0648
OPERATOR + TRUEJ 00064400 0649
PROCEDURE OPERATOR (HERE)J VALUE HEREJ ALPHA HEREJ BEGIN
OPERATOR + TRUEJ IF HERE ISN'T PLUS V MINUS V TIMES V 00064500 0650
OPERATOR + TRUEJ 00064600 0651
PROCEDURE OPERATOR (HERE)J VALUE HEREJ ALPHA HEREJ BEGIN
OPERATOR + TRUEJ IF HERE ISN'T PLUS V MINUS V TIMES V 00064700 0652
OPERATOR + TRUEJ 00064800 0653
PROCEDURE OPERATOR (HERE)J VALUE HEREJ ALPHA HEREJ BEGIN
OPERATOR + TRUEJ IF HERE ISN'T PLUS V MINUS V TIMES V 00064900 0654
OPERATOR + TRUEJ 00065000 0655
PROCEDURE OPERATOR (HERE)J VALUE HEREJ ALPHA HEREJ BEGIN
OPERATOR + TRUEJ IF HERE ISN'T PLUS V MINUS V TIMES V 00065100 0656
OPERATOR + TRUEJ 00065200 0657
SLASH V STAR V PERIOD V AMPERSAND V ARROW 00065200 0658
THEN OPERATOR = FALSE END OPERATOR;
00065300 0659
PROCEDURE BE (TRUETOG); VALUE TRUETOG I BOOLEAN TRUETOG I BEGIN
00065400 0660
LABEL FIN;
00065500 0661
IF REAL (TRUETOG) = C THEN INJECT (6) "6,NOT.(");
00065600 0662
ELSE UNIT = TRUE COMMENT JUST IN CASE THERE IS NO REL
00065650 0663
DO BEGIN
00065700 0664
IF HERE IS NOTW THEN BEGIN
00065800 0665
EMIT ("5,NCT.", 5); SCAN END;
00065900 0666
IF HERE IS PAREN THEN BEGIN
00066000 0667
EI SCAN; BE (TRUETOG));
00066100 0668
IF HERE ISN'T SPAREN THEN BEGIN
00066200 0669
GOTLOST (22); FLASH (SPAREN) END;
00066300 0670
EI SCAN;
00066400 0671
IF OPERATOR (HERE) THEN DO BEGIN
00066500 0672
EI SCAN; AE END
00066600 0673
UNTIL NOT OPERATOR (HERE) END
00066700 0674
ELSE AE;
00066800 0675
IF REAL (TRUETOG) = 2 THEN BEGIN
00066900 0676
IF HERE IS LESSTHAN OR HERE IS LEQ OR HERE IS
00067000 0677
EQUAL OR HERE IS NOTEQUAL OR HERE IS GTRTHAN
00067100 0678
OR HERE IS GEQ THEN BEGIN
00067200 0679
REL + HERE).(1216); EMIT (MINUS,1);
00067300 0680
UNIT = FALSE END;
00067400 0681
ELSE GO TO FIN END;
00067500 0682
ELSE IF HERE IS LESSTHAN THEN EMIT ("4.LT.",4)
00067600 0683
ELSE IF HERE IS LEQ THEN EMIT ("4.LE.",4)
00067700 0684
ELSE IF HERE IS EQUAL THEN EMIT ("4.EQ.",4)
00067800 0685
ELSE IF HERE IS NOTEQUAL THEN EMIT ("4.NE.",4)
00067900 0686
ELSE IF HERE IS GTRTHAN THEN EMIT ("4.GT.",4)
00068000 0687
ELSE IF HERE IS GEQ THEN EMIT ("4.GE.",4)
00068100 0688
ELSE IF HERE IS DLOW THEN EMIT ("4,LT.",4)
00068200 0689
ELSE IF HERE IS ANDM THEN EMIT ("5,AND.",5)
00068300 0690
ELSE IF HERE IS EQVW OR HERE IS IMPW THEN SCAR
00068400 0691
ELSE GO TO FIN;
00068500 0692
SCAN END
00068600 0693
UNTIL FALSE);
00068700 0694
FIN; IF REAL (TRUETOG) = 0 THEN INJECT (1,SPAREN)
00068800 0695
END REI;
00068900 0696
PROCEDURE AE;
00069000 0697
BEGIN
LABEL SAE;
00069100 0698
IF HERE IS IFW THEN BEGIN
00069200 0699
SCAR; EI SCAN; BE (TRUE);
00069300 0700
IF HERE ISN'T THENW THEN BEGIN
00069400 0701
GOTLOST (6); FLASH (THENW) END;
00069500 0702
TABSET; ENEWJ; SCAR; SCAN; AE;
00069600 0703
IF HERE ISN'T ELSEW THEN BEGIN
00069700 0704
GOTLOST (6); FLASH (ELSEW) END;
00069800 0705
ENEWJ; SCAR; SCAN; AE;
00069900 0706
TABLE CLEAR END ARITHMETIC IF SIDESTEP
00070000 0707
ELSE BEGIN
00070100 0708
IF HERE IS PLUS OR HERE IS MINUS THEN BEGIN
00070200 0709
EI SCAN END;
00070300 0710
SAE; TERM;
00070400 0711
IF HERE IS PLUS OR HERE IS MINUS THEN BEGIN
00070500 0712
PROCEDURE
PROCEDURE

STATEMENT FOR LIST;
ENLIST (SPECIES, TYPE);

INTEGER SPECIES, TYPE;
FORWARD;

AWRY (SPECIES, TYPE);
INTEGER SPECIES, TYPE;
BEGIN

BOOL(AN SAVESPACE;
SCAN;
IF HERE
IS COMMA
BEGIN
ALPHA ARRAY
AWRY ITEM (014J;
MOVE (5, INTOBUF, AWRYITEM .];
SCAN; ENLIST (SPECIES, TYPE);
I
INFOAUFR) MOVE (5, AWRYITEM[.]J, INTOAUFR)
END
ELSE IF HERE
IS
BEGIN
SCAN; DIM.
1;
If HERE IS "10
DO
BEGIN
GOTLOST (21)J
00076100
00076200
00076300
00076400
00076500
00076600
00076700
00076800
00076900
00077000
00077100
00077200
00077300
00077400
00077500
00077600
I F NEXT I
S: T
00 SCAN
SBRACKET
AND NEXT
ISNT COMMA THEN
00077700
00077800
00077900
I F NEXT I
S: T
00 SCAN
UNTIL HERE
IS COLON OR
NEXT
IS COMMA
OR
NEXT
IS SBRACKET;
I F HERE
IS COMMA
THEN DIM •
00078000
00078100
00078200
00078300
00078400
00078500
00078600
00078700
END; 00078800
00078900
00079000
00079100
00079200
00079300
00079400
00079500
00079600
00079700
00079800
00079900
00080000
00080100
00080200
I F NnT
BODLE
THE~
ELSE
BE (TRUE)
END ASSIr.NSTMT;

INTEGER STREAM PROCEDURE PASTPRIMAGy (H, P, C);
VALUE C;
RrGIN
LABEL AA, BA, CC;
SI • Pi SI. SI +
C;
SI • SI-l; % LAST
CHR
or
PRI~ARY
END ASSIr.NSTMT;
IF HERE IS "ELSE " THEN BEGIN
  EMIT (COMMA.1») SCAN; DOW ("DOW","1»)
END

IF HERE IS "4STEP " THEN BEGIN
  SCAN; PRIMARY) 00088100
END

IF HERE IS UNTILW THEN BEGIN
  DO BEGIN E; SCAN UNTIL HERE IS "5UNTIL 
END

ELSE BEGIN
  EMIT (COMMA.1») EMIT (INCREMENT. INN) END
END

ELSE BEGIN
  GOTLOST (3»)
END

IF HERE IS "5UNTIL " THEN BEGIN
  SCAN) PRIMARY) 00088900
  EMIT (DOW,3)) EMIT (SCOPE ») END
END

PROCEDURE FORSTMT; BEGIN
  ALPHA SCOPE, INCREMENT, INTEGER INN» SCAN;
  GLOBALCHECK;
  PRIMARY; ASSIGNSTMT;
  IF HERE IS "4STEP " THEN BEGIN
    EMIT (COMMA.1)) SCAN; IF HERE IS MINUS THEN GOTLOST(2)
    INCREMENT » HERE; IN » NU) SCAN;
    IF HERE » "5UNTIL " THEN BEGIN
      GOTLOST (IF HERE ISNT WHILEM THEN 24 ELSE 2)
    END
    DO BEGIN E; SCAN END UNTIL HERE IS "5UNTIL 
  END

  IF HERE IS "5UNTIL " THEN BEGIN
    SCAN) PRIMARY) 00089000
    EMIT ("1", ") SCAN;
    IF HERE _ "200 
  END

  IF HERE IS "5UNTIL " THEN BEGIN
    SCAN) PRIMARY) 00089000
    EMIT ("1", ") SCAN;
    IF HERE _ "200 
  END

  ELSE BEGIN
    GOTLOST (1»)
  ELSE BEGIN
    EMIT ("6) GO T", 6») EMIT ("20 ",2»)
    EMIT (SCOPE ») ENEW END;
  END

  SCAN END DOSTMT;

PROCEDURE DOSTMT; BEGIN
  ALPHA SCOPE, FIN;
  EMIT (SCOPE ») SCAN; STMT
  IF HERE » "5UNTIL " THEN BEGIN
    GOTLOST (2») FLASH ("DOW ");
  END

  IF HERE IS UNTIL THEN BEGIN
    ENEW) EMIT ("5IF ( ",4) SCAN;
    BE (BOOLEAN (IF OLDF THEN 2 ELSE 0)) ENEW OK DO;
    IF NOT TERMINATOR (HERE) THEN REMARK;
    IF OLDF THEN BEGIN
      OLDIFSTMT FIN» SCOPE») EMIT (FIN) END;
    END

    ELSE BEGIN
      EMIT ("6) GO T", 6») EMIT ("20 ",2»)
    END

    EMIT (SCOPE ») ENEW END;

  SCAN END DOSTMT;

PROCEDURE WHILESTMT; BEGIN
  ALPHA BRANCHBACK, BRANCHFORWARO, MEDIAS, H, N;
  EMIT (BRANCHBACK ») EMIT ("4IF ( ",4)
  SCAN; BE (BOOLEAN (IF OLDF THEN 2 ELSE 0))
  IF HERE » "200 " THEN BEGIN
    GOTLOST (3») FLASH ("200 ");
  END

  IF OLDF THEN BEGIN

- 71 -
PROCEDURE ESPACING (F, SP) VALUE F, SP; ALPHA F, SP; BEGIN
  EMIT (F) I
  EMIT (FORMAT, 6) I EMIT ("4 lH ", 4) I
  INJECT (1, SP); EMIT (SPAREN, 1) I
  ENDW EMIT (WRITE, 6) I EMIT ("3 (6, ", 3) I
  EMIT (F, 5) I
  EMIT (SPAREN, 2) I
END SPACING

PROCEDURE EFORMAT (F, SP) VALUE F, SP; ALPHA F, SP; BEGIN
  ALPHA FIRST I
  INTEGER NEST I
  NEST = 0 I
  IF HERE IS PAREN THEN NEST + NEST + 1 I
  ELSE IF HERE IS SPAREN THEN NEST + NEST - 1 I
  IF FIRST + HERE. (1216) = "X" THEN BEGIN
    IF HERE. (1216) = " " THEN HERE + "2 X " I
    & HERE. (1216) I
    ELSE HERE + "3 X " & HERE. (1218112) I
    END W
ELSE IF FIRST = "R" OR FIRST = "J" OR FIRST = "K" THEN BEGIN
  MARK (FIRST); END I
ELSE IF FIRST = "G" OR FIRST = "H" THEN BEGIN
  MARK (FIRST); END I
IF HERE IS GTRTHAN OR EXT AND HERE IS SPAREN THEN BEGIN
  NEST = 0 I
  IF SP = "10 " THEN EMIT (SLASH, 1); I
  IF SP = "11 " THEN EMIT ("4/1Hl ", 4); I
  EMIT ("1) ", I
  TABCLEAR; END W SCAN I
END EFORMAT.

PROCEDURE WRITESTMT (WRITE) VALUE WRITE, BOOLEAN WRITE; BEGIN
  ALPHA F, H; BOOLEAN DEFAULT, FORM, STUFF I
  INTEGER N, SPACING I
  NUSTMT = NUSTMT + 1 I
  SPACING + 0j DEFAULT + FORM + FALSE I
  SCANJ SCANJ % FIRST CHARACTER AFTER "(" I
  IF THIS IS SYM THEN BEGIN
    GET (HERE); IF IDTYPE = FILET THEN BEGIN
      SCANJ FORM + TRUE END I
    ELSE IF IDTYPE = OTHERT THEN GO TO WRITESTMT I
    ELSE BEGIN
      H + HERE; N + NU; DEFAULT + STUFF + TRUE I
      GO TO START END END I
    END I
    IF HERE IS "11 " THEN BEGIN
      SCANJ IF HERE IS PAGEM THEN SPACING + "11 " I
      ELSE IF HERE IS BALH THEN SPACING + "10 " I
      ELSE SPACING + HERE I
      SCANJ SCAN END I
      IF HERE IS "1) " THEN BEGIN
        ESPACING (F + NULL, SPACING); GO FAST END I
      IF HERE = "1) " THEN SCANJ I
      IF HERE IS "1k " THEN EFORMAT (F + NULL, SPACING) I
      ELSE IF FORM THEN BEGIN
        MATFIRST; F + FIND (HERE); SCAN END I
      ELSE BEGIN
        00098100 0988
        00098200 0989
        00098300 0990
        00098400 0991
        00098500 0992
        00098600 0993
        00098700 0994
        00098800 0995
        00098900 0996
        00099000 0997
        00099100 0998
        00099200 0999
        00099300 1000
        00099400 1001
        00099500 1002
        00099600 1003
        00099700 1004
        00099800 1005
        00099900 1006
        01000000 1007
        01000100 1008
        01000200 1009
        01000300 1010
        01000400 1011
        01000500 1012
        01000600 1013
        01000700 1014
        01000800 1015
        01000900 1016
        01001000 1017
        01001100 1018
        01001200 1019
        01001300 1020
        01001400 1021
        01001500 1022
        01001600 1023
        01001700 1024
        01001800 1025
        01001900 1026
        01002000 1027
        01002100 1028
        01002200 1029
        01002300 1030
        01002400 1031
        01002500 1032
        01002600 1033
        01002700 1034
        01002800 1035
        01002900 1036
        01003000 1037
        01003100 1038
        01003200 1039
        01003300 1040
        01003400 1041
        01003500 1042

- 73 -
PROCEDURE RACKSTRING;
BEGIN
FASTI SCAN;
STARTI 00103900
IF R THEN BEGIN
EMIT("6WRITE ",6) EMIT("3(6,",3) END 00104000
ELSE BEGIN EMIT("SPREAD ",5) EMIT("3(5,"3) END 00104100
IF F=0 THEN GOTLOST(5); EMIT(5)) EMIT(SPAREN+2)) 00104200
IF STUFF THEN BEGIN
IF DEFAULT THEN BEGIN
HERE + H; NU + N END
ELSE SCAN;
END;
ELSE STUFF;
IF HERE IS NOT SPAREN THEN BEGIN
GOTLOST(17); FLASH("1") END END;
FASl SCAN;
IF HERE IS "II" THEN BEGIN
GOTLOST(18); FLASH(SBRACKET) END;
IF NOT TERMINATOR(HERE) THEN BEGIN
SCAN;
END;
SCAN;
END;
FALSE;
END;
FALSE;
END 00103700
00103800
00103900
00104000
00104100
00104200
00104300
00104400
00104500
00104600
00104700
00104800
00104900
00105000
00105100
00105200
00105300
00105400
00105500
00105600
00105700
00105800
00105900
00106000
00106100
00106200
00106300
00106400
00106500
00106600
00106700
00106800
00106900
00107000
00107100
00107200
00107300
00107400
00107500
00107600
00107700
00107800
00107900
00108000
00108100
00108200
00108300
00108400
00108500
00108600
00108700
00108800
00108900
00109000
00109100
00109200
00109300
00109400
00109500
00109600
00109700
END TYPE CASES;

END;

COMMENT ARRAYS; BEGIN
  ARRAY (SPECIES, TYPE); CHCKINFO (LENGTH + 7*DIM);
  IDARRAYBITS + DIM; Go TO CHAIN END;
COMMENT LABELS; BEGIN
  IDLAB + NUL; TYPE + LABELT; LENGTH + 5
END LABELS;

COMMENT 4--PROCEDURES; BEGIN
  IF NOT SPECIFICATIONS THEN PRO (TYPE); IMPROBIT + 100110000
  LENGTH + 5
  IF TYPE ≠ 0 THEN GO TO CHAIN END;
COMMENT 5--OWNS; BEGIN
  PUTINCOMMON; LENGTH.
END

COMMENT 6--SWITCHES; BEGIN INTEGER HI;
  SCAN; COMMENT TO GET;
  DO BEGIN
    SCAN; GET (HERE); INFO [15, HI+HI+1] + IDLAB;
    SCAN END
UNTIL HERE IS SEMICOLON; I + INFOBUFFER;
  IDARRAYBITS + (LENGTH + HI+1) - 4
END SWITCHES;

COMMENT 7--DEFINES; BEGIN
  SCAN;
  CHECKINFO (LENGTH, RACKSTRING • 5);
END;

COMMENT 8--FORMATS; BEGIN
  SCAN;
  EXT. TRUE;
  EFORMAT (IDLAB + NUL, 0); EXT + FALSE;
  LENGTH + 5 END FORMATS;

COMMENT 9--FILES; BEGIN
  LENGTH + 5;
  FLASH (SPAREN);
END FILES;

COMMENT 10--LISTS; BEGIN
  FLUSH;
END LISTS;

END SPECIES CASES;

COMMENT ALL THE IO WORDS ARE FULL NOW EXPECT THE FIRST;
  CHECKINFO (LENGTH);
  INFORUF.LINKB • INFO [0,INDEX];
  INFO [0,INDEX] + NEXTINFO;
  MOVE (LENGTH, INFOBUF, INFOIN);
  NEXTINFO + NEXTINFO + LENGTH;
  IF NEXT IS SEMICOLON OR NEXT IS COMMA THEN SCAN;
END ENLIST;

PROCEDURE DECLARATIONS; BEGIN
  LABEL SMFLIL, SMFORMAT, SWLIST, SP, OTHERKIND, ARRAYL00113700
  BOOLEAN MAKE, SHORT; INTEGER SPECIES, TYPE;

DO BEGIN
  MAKE + SHORT + TRUE;
  IF HERE IS SAVEN THEN SCAN;
  IF HERE IS OWNS THEN BEGIN
    SPECIES + OWNS; SCAN END;
  IF HERE IS REALW THEN TYPE + REALT
  ELSE IF HERE IS INTEGERW THEN TYPE + INTEGER
      0014400 0014400

    0014300 0014300

    0014200 0014200

    0014100 0014100

    0014000 0014000

    0013900 0013900

    0013800 0013800

    0013700 0013700

    0013600 0013600

    0013500 0013500

    0013400 0013400

    0013300 0013300

    0013200 0013200

    0013100 0013100

    0013000 0013000

    0012900 0012900

    0012800 0012800

    0000000 0000000

      - 75 -
ELSE IF HERE IS BOOLEAN THEN TYPE + BOOLEAN
ELSE IF HERE IS ALPHANW THEN TYPE + REAL
ELSE IF HERE IS ARRAYW THEN BEGIN
    TYPE + REALT; GO TO ARRAYL END
ELSE GO TO OTHERKIND;
SCAN;
IF HERE IS STREAM THEN BEGIN
    GOTLOST (20); FAMSTRAY + TRUE; SCAN ENDS; XOR
ELSE IF HERE IS PROM THEN SPECIES + PROS
ELSE IF HERE IS ARRAYW THEN ARRAYL; SPECIES + ARRAYS
ELSE BEGIN
    SHORT + FALSE; SPECIES + IDS ENDS;
ELSE GO TO OTHERKIND;
    TYPE + OTHERT;
ELSE BEGIN
    IF MAKE
        THEN BEGIN
            IF SHRT
                THEN SCAN;
            I = INFOBUFFER;
            COMMENT NOW WE ARE LOOKING AT A LIST OF ID"S;
            IDBLOCK + BLOCKCOUNT; IDTYPE + TYPE
            ENLIST (SPECIES, TYPE);
            WHILE HERE IS CCMMA DO BEGIN
ALPHA PROCEDURE VFORM (X); VALUE X; ALPHA X; VFORM + O & (X, [616]+1) [614216] & "V" [1214216]
& X [18112130];

ALPHA STREAM PROCEDURE COMNAME (OUTER, INNER); VALUE OUTER, INNER; BEGIN
DI + LOC COMNAMEJ DI + DI + 4J SI + LOC INNERJ
SI + 2 DEC SI + LOC OUTERJ DS + 2 DEC END COMNAMEJ
PROCEDURE ADOTHECOMMONBLOCK (NEW) "TO THE BLOCK" (BC); VALUE NEW;
ALPHA NEW, BC, BEGIN
IF BC.COMN _ NEW THEN
IF BC.LINKB _ 0 THEN
ADOTHECOMMONBLOCK (NEW, INFO[BC.RB, BC, NB])
ELSE BEGIN
INFO.COM + NEW; NIF 1 END
END ATTHECOMMONBLOCK;
PROCEDURE PUTINCOMMON; BEGIN
ALPHA NEWCOM, INTEGER INDEX, OUTERBC;
NEWCOM + COMNAME (IDBLOCK, BLOCKCOUNT); IF IDCOM = 0 THEN BEGIN INTOCOM + INTOCOM + 1;
IF CID ISNT NEWCOM THEN BEGIN
IF CID _ 0 THEN MOVE (6, CLINK, INFOCOMINDEX); COMMENT PUT CURRENT COMITEM INTO ITS INFO LOC;
LOAD (NEWCOM); IF CID = 0 THEN BEGIN
CID + NEWCOM CHECKINFO (6); CLINK + INFO (0, INDEX, SCRAMBLE (NEWCOM)); INFO[0, INDEX] + COMINDEX + NEXTINFO;
NIF 6; NUCCOM + NUCCOM + 1 END;
END NEW COMITEM;
IDCOM + CID;
CASE IOTYPE OF BEGIN
BEGIN IDCCOMLINK + CR; CR + I+6 END;
BEGIN IDCCOMLINK + CI; CI + I+6 END;
BEGIN IDCCOMLINK + CR; CB + I+6 END;
BEGIN IDCCOMLINK + CO; CD + I+6 END;
END LINKING ITEM TO COMMON BLOCK TYPE5;
ADOTHECOMMONBLOCK (CID, BC);
END ZERO IDCOM
ELSE BEGIN IF IDCOM ISNT NEWCOM THEN BEGIN
ADOTHECOMMONBLOCK (IDCOM, BC); NEWCOM + IDCOM END;
END
OUTERBC + BLOCKSTACK (IDBLOCK)
ADOTHECOMMONBLOCK (NEWCOM, INFO [OUTERBC, RP, OUTERBC, NB]);
END PUTINCOMMON;
%---------------------------------------------------------------
% PRINCIPAL PROCEDURES
%---------------------------------------------------------------
PROCEDURE SAVEBLOCK (PAST); INTEGER PAST; BEGIN

PROCEDURE MOVE (5, BC, INFOLINDEX);
PAST + BLOCKCOUNT;
IF NEXTBLOCK + (BLOCKCOUNT + NEXTBLOCK) + 1 = 100
THEN GOLOST (7);
CHECKINFO (5); BLOCKINDEX + NEXTINFO; NIF 5;
BC + Bi + BR + BO + OJ TABSET; TYPEWRITE;
END;

PROCEDURE CUTBACK; BEGIN
    INDEX, I, J;
    FOR INDEX = 0 STEP 1 UNTIL INFORMLENGTH - 1 DO
        IF J = INFO [I, INDEX] = 0 THEN BEGIN
            BEGIN I = JJ; J = STARTLINK END
        UNTIL IDDBLOCK = BLOCKCOUNT;
    INFD [0, INDEX] = I END;
END CUTBACK;

PROCEDURE RESTOREBLOCK (PAST); BEGIN
    INJECT (6, "RETURN"); ENWW; INJECT (3, "END "); ENWW;
    BLOCKRETR + BLOCKRETR + 1;
    IF BLOCKCOUNT > 1 THEN CUTBACK;
    MOVE (5, BC, INFOLINDEX);
    BLOCKCOUNT + PAST;
    MOVE (5, INFOLINDEX, BC);
END RESTOREBLOCK;

PROCEDURE PRO (TYPE); BEGIN
    ALPHA H; BOOLEAN CMPDTAIL, VALUEPARAMETERS;
    LAREL FUNNYCOMMA; INTEGER PASTBLOCK, F, r;
    ALPHA ARRAY FORMAL [0:25], PROITEM [0:15];
    MOVE (5, STACKLINK, PROITEM [0]);
    SAVEBLOCK (PASTBLOCK);
    IF CC THEN BEGIN
        COMMENT 7090 DECK HEADING;
        FILL PIFFLE [*] WITH "$IBHTC A=9999 NODECK";
        HERE + DECKNAME (BLOCKCOUNT); SEND (6, HERE, PIFFLE [0], 7);
    ENWW END HEADING;
    IF TYPE = 0 THEN BEGIN
        EMIT ("SUBROU"), 6; EMIT ("STINE "), 5 END
    ELSE BEGIN
        EMIT ("SUBROU"), 6; EMIT ("STINE "), 5 END
    ELSE IF TYPE IS REAL THEN BEGIN
        EMIT ("REAL "), 5 END
    ELSE IF TYPE IS INTEGER THEN BEGIN
        EMIT ("INTEGER "), 6 END
    ELSE BEGIN
        EMIT ("LOGICAL "), 6; EMIT ("INTE "), 2 END
    END
    IF HERE IS PAREN THEN BEGIN
        EMIT (PROITEM [1]); SCAN;
        EMIT (PROITEM [1]); SCAN;
    IF HERE IS SPAREN THEN BEGIN
        EMIT (PROITEM [1]); SCAN;
        EMIT (PROITEM [1]); SCAN;
    IF HERE IS VALUE THEN BEGIN
        EMIT (PROITEM [1]); SCAN;
        EMIT (PROITEM [1]); SCAN;
    END
    IF HERE IS VALUE THEN BEGIN
        EMIT (PROITEM [1]); SCAN;
        EMIT (PROITEM [1]); SCAN;
    END
END;

- 78 -
VALUEDPARAMETERS + TRUE
DO BEGIN
  SCAN; FOR G+O STEP 1 UNTIL F DO
  IF FORMAL [G], (211) = 0 THEN BEGIN
    FORMAL [G], (211) + 1
    G + F END
  SCAN END
  UNTIL HERE IS SEMICOLON
  SCANJ
END VALUE LIST
ELSE VALUEDPARAMETERS + FALSE
COMMENT S AVE FIRST WORD OF BODY OR HEADING; HERE;
COMMENT EMIT FORMALS; FOR G+O STEP 1 UNTIL F DO BEGIN
  EMITN (IF FORMAL [G], (211) = 0 THEN FORMAL [G],)
  ELSE VFORM (FORMAL [G]));
  IF G < F THEN EMIT (COMMA,1) ENDJ
EMIT (SPAREN,1) ENEW;
HERE + Hj SPECIFICATIONS + TRUE
DECLARATIONS;
H + HEREj SPECIFICATIONS + FALSE
IF VALUEDPARAMETERS THEN BEGIN
  BOOLEAN NONE;
  INTEGER K;
  FOR K = REAlT, INTEGER, BOOLEAN, OTHERT
  DO BEGIN
    NONE, TRUE;
    FOR G+O STEP 1 UNTIL F DO
    IF FORMAL [G],(61421) = 1 THEN BEGIN
      GET (FORMAL [G],(61421));
      IF IDTYPE = K THEN BEGIN
        IF NONE THEN BEGIN
          NONE, FALSE;
          IF K = REAlT THEN
            EMIT ("5REAl ",5)
          ELSE IF K=INTEGER THEN BEGIN
            EMIT ("6INTEG",6);
            EMIT ("5R ",2) END
          ELSE IF K=BOOLEAN THEN BEGIN
            EMIT ("6LOGICA",6);
            EMIT ("2R ",2) END
          ELSE IF K=OTHERT THEN BEGIN
            EMIT ("5FORMAT",5);
            EMIT ("2R ",2) END
          END
        ELSE EMIT (COMMA,1) END
      END
      ELSE EMIT (COMMA,1)
    END
    EMIT ("8REAL ",5)
  END
END DECLARING FORMAL PARAMETERS;
HERE + Hj DIFFERENT TYPES OF FORMALS
END DECLARING FORMAL PARAMETERS;
ELSE BEGIN
  ENEW; SCAN EN EW PARAMETERLESS PROCEDURE;
END
IF VALUEOPARAMETERS THEN
  FOR G=0 STEP 1 UNTIL F DO
    IF FORMAL [G].[21] = 1 THEN BEGIN
      EMITN (FORMAL [G])
      INJECT (1*EQUAL)
      EMITN (FORMAL [G]2)
      ENEW ENEW
    END BEGIN
    MOVE (S, PROITEM[*], INFOBUF)
    I = INFOBUFFER
    RESTOREBLOCK (PASTBLOCK)
    EAMSTRAY = FALSE
  END
PROCEDURE STMT BEGIN
  WHILE NEXT IS COLON DO NUMBER
  IF NEXT IS ARROW THEN BEGIN
    GLOBALCHECK
    NUSTMT + NUSTMT + 1
    IF IDTYPE = BOOLEAN THEN BOOLEAN + TRUE
    EJ SCAN
    ASSIGNSTMT
    BOOLEAN + FALSE END
  ELSE IF HERE IS BEGINW THEN BEGIN
    SCAN
    IF DECL THEN BLOCK ELSE
      COMPOUNDTAIL
  END
  ELSE IF HERE IS IFW THEN IFSTMT
  ELSE IF HERE IS FORW THEN FORSTMT
  ELSE IF HERE IS GOW THEN GOTOSTMT
  ELSE IF HERE IS ENDW THEN ENEW
  ELSE IF HERE IS DOW THEN OOSTMT
  ELSE IF HERE IS WHILEW THEN WHILESTMT
  ELSE IF NEXT IS PAREN THEN BEGIN
    IF HERE IS WRITETHEN WRITESTMT (TRUE)
    ELSE IF HERE IS READ THEN WRITESTMT (FALSE)
    ELSE IF HERE IS "RELEAS" OR HERE IS "CLOSE" OR
    HERE IS "SPACE " OR HERE IS "REWIND" OR
    HERE IS "LOCK" OR HERE IS "DOUB" THEN BEGIN
      MARK ("*") FLUSH END
    ELSE IF EAMSTRAY THEN NESTSTMT
    ELSE ECALL END NEXT IS PAREN
  ELSE IF HERE IS FILMW THEN FILLSTMT
  ELSE IF HERE IS ELSEW THEN FLUSH
  ELSE IF THIS IS SYM THEN BEGIN
    IF HERE IS JUMPW THEN JUMPSTMT ELSE
    IF NEXT IS BRACKET OR NEXT IS PERIOD THEN BEGIN
      GLOBALCHECK
      IF IDTYPE = BOOLEAN THEN BOOLEAN + TRUE
      PRIMARY
      IF HERE IS ARROW THEN BEGIN
        NUSTMT + NUSTMT + 1
        ASSIGNSTMT
        BOOLEAN + FALSE END
      ELSE BEGIN GOTOSTMT (19); EST END END
    ELSE IF NEXT IS SEMICOLON THEN ECALL
    ELSE BEGIN DECLARATIONS
      IF NOT TERMINATOR (HERE) THEN BEGIN
        FLASH (END); IF HERE ISN'T ENDW THEN SCAN
      END BEGIN
    END
INTEGER BLOCKCOUNT, BEGIN  
INTEGER L, CI, ALPHA, COMID, CTY;
MARGIN + PIFFLE (11) TYPEWRITE;
MOVE (10, FRESH(*, PIFFLE(*)*));
MOVE (5, INFOBLOCKINDEX, BC) GROUP + TRUE;
IF BC ≠ 0 THEN BEGIN
  L = INFO [BC, RB, BC, NB] END;
DO BEGIN
  COMID + L, COM & "CM" [1213612];
  LOAD (L.COMN);
  EMIT ("6COMMON", 6); EMIT ("2 /", 2);
  EMIT (COMID, 6); EMIT ("2/", 2);
  FOR CTY = CR, CI, CB DO
    IF CTY.LINKB ≠ 0 THEN BEGIN
      DECLARE (CTY, 6) GROUP. NONE
      END;
    END;
  EMIT ("5REAL", 5) DECLARE (CR, 6) ENEW;
  IF CR.LINKB ≠ 0 THEN BEGIN
    EMIT ("6INTEGER", 6) DECLARE (CR, 6) ENEW;
    END;
  IF CI.LINKB ≠ 0 THEN BEGIN
    EMIT ("6INTEGER", 6) DECLARE (CI, 6) ENEW;
    END;
  IF CB.LINKB ≠ 0 THEN BEGIN
    EMIT ("6LOGICAL", 6) DECLARE (CB, 6) ENEW;
    END;
  END COMMON DECLARATIONS;
  IF BR ≠ 0 THEN BEGIN
    EMIT ("5REAL", 5) DECLARE (BR, 4) ENEW;
    END;
  IF BI ≠ 0 THEN BEGIN
    EMIT ("6INTEGER", 6) DECLARE (BI, 4) ENEW;
    END;
  IF BB ≠ 0 THEN BEGIN
    EMIT ("6LOGICAL", 6) DECLARE (BB, 4) ENEW;
    END;
  END DECLARATIVE STATEMENTS;
PROCEDURE POSTMORTEM;
BEGIN
  FORMAT OUT SUMMARY (J " INPUT CARDS WERE TRANSLATED, "
    "PRODUCING "J" OUTPUT CARDS, AN INCREASE OF "J" /
    "20 J " OF THE OUTPUT CARDS WERE FLAGGED, BATTLING "
    "AVERAGE "F5.3 / " / "ELAPSED TRANSLATION TIME = "
    "F6.2 " SECONDS" / ");
  INFOSTAT ("INFO ABSORBED "J" IDENTIFIERS, WHICH USED "J"
    "WORDS, THEY WERE USED "J" TIMES IN "J" STMTS ("J"
    "STMT") / ");
  IDTEW (J " SCANS ITEMS/STMT "J);
  COMSTAT (J " COMMON BLOCKS "J")
    COMMON BLOCKS ("J" "BLOCK")/;
  RETRSTAT ("RETRIEVED " IDENTIFIERS "J" BLOCKS "J"
    "COMMON "J");
  FINAL ("TIME OFF "J")
  WRITE (OUTFIL [PAGE]), DATEI;
  WRITE (OUTFIL, SUMMARY, INPUT, TOTAL (TOTAL-INPUT)/INPUT
    "X100 "BAD, BAD/BAD(TOTAL, (TIME(2)-SECONDS)/60);
  WRITE (OUTFIL, INFOSTAT, ENLISTED, NEXTINFO=256,
    00152200) 1537
NUPRIM, NUSTMT, NUPRIM/NUSTMT)
WRITE (OUTFIL, IOTEM, IOTA, IOTA/NUSTMT)
IF NUCOM NE 0 THEN WRITE (OUTFIL, COMSTAT, INTOCOM, ENTER (100XINTCOM/ENLISTED), NUCOM, ENTER (INTCOM/NUCOM))
WRITE (OUTFIL, RETRSTAT, IDRETR, BLOCKRETR, COMRETR)
END POSTMORTEM
%------------------------------------------
% INITIALIZATION
%------------------------------------------
LABEL ALLGONE
FILL FRESH [*] WITH ""
FILL DIRECTIVE [*] WITH ZIP WITH DIRECTIVE [*] SECONDS. TIME(2) WRITEDISK = TRUE

MOVE (10, FRESH [*], PIFFLE)

COLUMN + 1; THIS *SCHR* DO SCAN UNTIL TABLE [0] ISNT O; IF HERE ISNT BEGIN THEN BEGIN PLACE + 72; DO BEGIN ECOMMENT* SCAN END INITIAL REMARKS UNTIL HERE IS BEGIN END FINDING THE FIRST BEGIN; IF CC THEN BEGIN FILL PIFFLE* WITH "$$JOB GO = FIOCS $ENEM END$" 00152900 1538

RETURN (PARTIAL);
WRITEDISK = FALSE
DO BEGIN
READ (PARTIAL, 15, PIFFLE[*]) (ALLGONE)
IF PIFFLE [12] NE 0 THEN DECLARATIVESTATEMENTS (PIFFLE [12]); ELSE ENEM* COMMENT ENEM* (WITH WRITEDISK = FALSE) PRODUCES A LINE OF OUTPUT ON THE PRINTER OR PUNCH;
END
UNTIL FALSE) ALLGONE: WRITE (OUTFIL [PAGE]);
POSTMORTEM
END.