A Token Recognizer For The Standard Hardware Representation of Algol 68.

by

R. Bell, Department of Computer Science,
Teesside Polytechnic,
Middlesbrough,
England.

Introduction

This token-recognizer is designed to scan texts which purport to be Algol 68 particular-programs in the standard hardware representation defined by Hansen and Boom 1. It will seek to parse each given text into a sequence of language tokens, digestible by, for instance, the syntax analyser of an Algol 68 compiler.

As the word "token" bears a specialized meaning in Algol 68, this document will instead speak of "words", which are, broadly Algol 68 TAX-symbols, denotations or other NOTION-symbols 2. Each activation of the recognizer will deliver a representation of just one such "word" to the superior routine that drives it.

This recognizer may serve, it is hoped, as a general purpose front-end component, not only for full compilers but also for syntax checkers or preprocessors.

The algorithm is presented in Algol 68. Readers are warned that it has not been machine-checked directly (because the author has no access to any compiler for canonical Algol 68). However, an analogous program in Algol 68R has been written and compiled and is being tested.

1 Hansen W.J. and Boom H., Report on the Standard Hardware Representation for Algol 68, (AB 40.5) in Algol Bulletin 40 (pp 24 - 43), 1976. (hereinafter designated by "HR").

The other fundamental document is, of course:


2 In this document, Algol 68 paranotions are hyphenated where necessary and (except in section 2) underscored.
1 Words

This recognizer does not deal with the following contexts in particular-programs:

(a) interiors of fragments (and by implication, their terminators);
(b) interiors of format-texts (and by implication, their terminators), except that it is applicable to closed-clauses, CHOICE-clauses, units or denoters discovered inside format-texts.

This recognizer may encounter, where it is applicable, six classes of "words". The initial character of a word implies its class.

It is assumed here that the set of "base characters" which occur in texts is identical to the set of "worthy characters" defined in HRI, and may include both upper and lower case letters.

The six classes of words are:

(1) Tags, i.e. TAG-symbols, which are identifiers, label-identifiers or field-selectors;

(2) Bold-words:

There are 61 specified bold-words which are fixed as representations of certain NOTION-symbols (see Appendix). Any other bold-word must be a bold-TAG-symbol, and as such either a mode-indication (TAB-symbol) or an operator (TAO-symbol);

HR3,5 explains how tags and bold-words are differentiated: mainly by "stropping", of which there are three alternative standard regimes, "point", "upper" and "res".

(3) integer-denotations, real-denotations, bits-denotations (also digit-symbols in priority-definitions);

(4) character-denotations, string-denotations;

(5) operators which are not bold-TAG-symbols, i.e. DOP-BECOMESBY-symbols; also the is-defined-as-symbol;

(6) Some other NOTION-symbols (e.g. $, |, : ).

Outside character- and string-denotations, "point" and "res" stropping do not distinguish between upper and lower case letters (a and A are regarded as the same character); "upper" stropping does distinguish (indeed, requires both cases to be used), and confines upper case letters to bold-words.
The classifying powers of initial characters of words are as follows:--

<table>
<thead>
<tr>
<th>CHARACTER</th>
<th>SIGNIFICANCE</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>a letter</td>
<td>&quot;point&quot; stropping : start of a tag</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>&quot;upper&quot; stropping : start of a tag</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>lower case letter : start of a tag</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>upper case letter : start of a bold-word</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>&quot;res&quot; stropping : start of a tag</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>or of a reserved bold-word</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>or of a tag followed by a reserved bold-word</td>
<td>1, 2</td>
</tr>
<tr>
<td>. (point)</td>
<td>if followed by a letter : start of a bold-word</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>if followed by a digit : start of a real-denotation</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>otherwise an incorrect character at this level</td>
<td></td>
</tr>
<tr>
<td>a digit</td>
<td>start of an integer- or real- or bits-denotation, or a digit-symbol (priority)</td>
<td>3</td>
</tr>
<tr>
<td>&quot; (quote)</td>
<td>start of a character- or string-denotation</td>
<td>4</td>
</tr>
<tr>
<td>% + - &lt; &gt; / *</td>
<td>start of an operator</td>
<td>5</td>
</tr>
<tr>
<td>=</td>
<td>start of an operator, or is-defined-as-symbol</td>
<td>5</td>
</tr>
<tr>
<td>: (colon)</td>
<td>label- or colon- or up-to- or routine-symbol, or start of becomes- or is- or isnt-symbol</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>(stick)</td>
<td>brief-then/in//else/out-symbol or start of brief-elseif/outline-symbol</td>
</tr>
<tr>
<td>= @ ( ) , ; @ [ ]</td>
<td>various NOTION-symbols</td>
<td>6</td>
</tr>
<tr>
<td>' -</td>
<td>incorrect characters at this level</td>
<td></td>
</tr>
</tbody>
</table>

Spaces and newlines are of no significance at this level. Logical-end-of-text might be treated as a fault, as Algol 68 particular-programs are supposed to be well-closed.
2 Algorithm

This is presented in an "upper stropped" representation of Algol 68, except that, as in RRIO, there are certain particular constructs whose precise forms are left to the discretion of implementors: these are informally described by "pseudo-comments" which are bounded by the marks $C ... C$.

The algorithm is given in two parts: the recognizer procedure, called "get word" and (preceding "get word") declarations necessary to create the environment for "get word".

Two details of the algorithm should be particularly noted.

Under "res" stropping it may be found that a reserved bold-word follows a tag. This possibility must be resolved during one activation of the recognizer: the tag is delivered and the reserved bold word is held in a (non-local) variable until it (and no subsequent word) is delivered on the next activation of the recognizer.

Certain concatenations of characters starting with DOP-symbols are ambiguous until more information about the context is known (which the recognizer in itself cannot provide). In concatenations such as $<=$, $<=$:=, the final "=" might be part of the operator or a separate is-defined-as-symbol. The latter is the case if it is a defining occurrence of the operator (i.e. in an operation-definition or a priority-definition and the next "word" is not also "=". All these ambiguous concatenations are split into two words by the algorithm.
{ 2.1 Environment }

COMMENT The following declarations are to be made in ranges embracing the declaration of the recognizer procedure 1: Forms dealing with character classification, cf HR C6

COMMENT

INT upletter = max abs char + 1,
adigit = max abs char + 2,
another = max abs char + 3;

[ : ]INT chartype
   =
      A row of integers with bounds [0 : max abs char],
      having the property implementation-dependent#

      chartype[i] =
         IF REPR i is neither a letter nor a digit
            THEN another
         ELIF REPR i is a digit
            THEN adigit
         ELIF REPR i is an upper case letter
            THEN upletter
         ELSE #(REPR i is a lower case letter)#
            ABS the corresponding upper case letter
         FI

PROC(REF CHAR)BOOL uletter = (REF CHAR c)BOOL :
   chartype[ABS c] = upletter ,
   sletter = (REF CHAR c)BOOL :
      IF INT ti = chartype[ABS c] ;
         ti <= max abs char
      THEN
         #( c refers to a lower case letter, which is replaced by
         the corresponding upper case letter )#
         c := REPR ti ;
         TRUE
      ELSE
         FALSE
      FI ;

PROC(REF CHAR)BOOL letter = (REF CHAR c)BOOL : uletter OR sletter ;
PROC(CHAR)BOOL digit = (CHAR c)BOOL : chartype[ABS c] = adigit ;
STRING emptystring = "" ,
CHAR underscore = " _ " , space = " " , quote = """" ,
apostrophe = C The denotation of the apostrophe character C ;
COMMENT 2: Forms dealing with reading the input text

COMMENT

REF CHAR char = LOC CHAR := space
#(to hold the character in hand)#,

REF BOOL eol = LOC BOOL := FALSE  #(see below)# ;

PROC(REF CHAR)VOID get next character
  = (REF CHAR ch)VOID
    C
    A routine which reads the next available character from the
    input text and assigns it to ch
    ( and perhaps also transcribes the input text to a listing
    ( into which warning and fault messages etc may be
    interpolated ) ).

Event routines for whichever file is currently accessing the
input text should behave as follows :-
  (a) On logical file end - resort to the operating-system,
    which may either (if commanded and able to)
    mend the file so that reading can continue from
    another input text (book)
    and make eol (see above) := TRUE ,
    or abort the run ;
  (b) On page end - call newpage and make eol := TRUE ;
  (c) On line end - call newline and make eol := TRUE

  #(hence if an event occurs and is cleared,
  eol = TRUE and the character from the next good position
  is assigned to ch )#
  C ;

#("point" stropping will be the default regime;
Stropping regimes are switched by pragmats, see HR3.5 )#

REF BOOL upperstrop = LOC BOOL := FALSE ,
resstrop = LOC BOOL := FALSE ;

# If the fixed-point-numeral of an INTREAL-denotation is followed
by a point, it is necessary to look ahead to see if the point is
followed by a letter, in which case INTREAL- is integer- and the
point must be deemed to be the strop for a following bold-word  #

REF BOOL intpointletter = LOC BOOL := FALSE ;
3: Forms associated with information generated by the recognizer

Each time it is called the recognizer generates a "word", which is a structured value consisting of a string and a procedure. The procedure will depend on what the word is that has been recognized in the input, and on the use to which the recognizer is being put.

The routines to be ascribed to these procedures are therefore left undefined here; provision is made for these routines to have parameters various in numbers and modes, by proposing that all the "word" procedures have one parameter whose mode is a union of a sufficient set of modes (left undefined here)

MODE WORDPARAMS = UNION ( C of a sufficient set of modes C ) ;

MODE WORD =
STRUCT ( STRING repstring , PROC(WORDPARAMS)VOID wordproc ) ;

PROC(WORDPARAMS)VOID

C definitions of procedures with the following identifiers :-

atproc, boldbeginproc, bitsmodeproc, .......
...... and similarly for all the reserved bold words ..... 
...... unionproc, voidproc, whileproc,

# and #

bouldtagproc, tagproc, bitsdenproc, badbitsdenproc,
inidenproc, realitenproc, badrealidenproc, chardenproc,
stringdenproc, estringdenproc, tadproc, taoproc,
bwordproc, equalsproc, colonproc, becomesproc,
badwordproc, briefwhelseoutproc, briefelifouseproc,
hashcommentproc, formatterproc, lpparenproc, rpparenproc,
andalsoproc, goonproc, briefsubproc, briefbusproc,
badcharproc

C ;

COMMENT
In two instances (as will be seen) the recognizer has to look one word ahead in the input text

COMMENT

REF BOOL word held = LOC BOOL := FALSE ,
REF WORD held word = LOC WORD ;
2.2 Recognizer Routine

PROC (REF WORD) VOID get word  
= (REF WORD w ) VOID  
: w :=  
IF word held  
THEN  
word held := FALSE ;  
held word  
ELSE  
  
# read and ignore any typographical features  
preceding a word #  
WHILE char = space  
DO get next character (char) OD ;  
eol := FALSE ;  
IF  
  
#1#  
BOOL ul = uletter(char),  
ll = sletter(char),  
dgt = digit(char),  
pt = char = "."  
# only one of these can be TRUE # ;  
IF pt THEN get next character (char) FI ;  
BOOL ptsameline = pt AND NOT eol  
OR intpointletter ;  
intpointletter := FALSE ;  
BOOL pul = ptsameline AND uletter(char) ,  
pill = ptsameline AND sletter(char) ,  
pdgt = ptsameline AND digit(char)  
# and only one of these can be TRUE # ;  
pt AND NOT( pul OR pill OR pdgt )  
THEN  
  
#1#  
C emit a fault message (impermissible character) C ;  
("." , badcharproc )  
ELIF  
  
#1#  
ul OR ll OR pul OR pill
THEN  #1#  
# a bold word or a tag #
PROC (PROC VOID) VOID break in tag
   = (PROC VOID p ) VOID
   : WHILE
      BOOL le = eol ;
      eol := FALSE ;
      BOOL u = char = underscore ;
      IF u
      THEN emit a warning
           (unwanted underscore in tag) C
           FI ;
      IF u OR char = space
      THEN get next character (char) ; TRUE
      ELSE le
      FI
   DO p OD ;

BOOL pointstrop = NOT (upperstrop OR resstrop) ;
IF  #2#
   pointstrop AND NOT pt OR upperstrop AND ll
THEN  #2#
   # a tag (for tags under resstrop see later) #
PROC (PROC(REF CHAR)BOOL) WORD tagscanner
   = (PROC(REF CHAR)BOOL charbool ) WORD
   : BEGIN
      REF STRING tagstring = LOC STRING := char ;
      WHILE
         get next character (char) ;
         IF NOT eol
            AND char = underscore
         THEN get next character (char)
         FI ;
         break in tag (VOID:SKIP) ;
         # one underscore is allowed after each taggle,
         newlines and spaces between
         taggles are immaterial #
         charbool (char)
         DO
            tagstring PLUSAB char
         OD ;
      ( tagstring, tagproc )
   END ;
IF upperstrop
THEN
tagscanner ((REF CHAR c)BOOL
  : sletter(c) OR digit(c) )
ELSE
tagscanner ((REF CHAR c)BOOL
  : letter(c) OR digit(c) )
FI

ELSE #2#

# a bold word if pointstrop or upperstrop,
either (or both) if resstrop #

PROC (STRING, REF WORD) BOOL matchres
  = (STRING charstring, REF WORD rword) BOOL
  : BEGIN
    # tests if charstring matches any
    reserved bold word #
    [ : ]WORD restable
      = ("AT", atproc
        ,"BEGIN", boldbeginproc
        ,"BITS", bitsmodeproc
        ,"C" ...... and so on
        for all the reserved
        bold words ..... 
        "UNION", unionproc
        ,"VOID", voidproc
        ,"WHILE", whileproc );

    [ : ]STRING resstrings
      = repstring OF restable ;
    INT top = UPB resstrings ;
    STRING firstres = resstrings[1]
    ,lastres = resstrings[top] ;
    REP BOOL found = LOC BOOL := FALSE ;
    IF charstring>=firstres
    AND charstring<=lastres
    THEN
      REF INT 1 = LOC INT ;
      IF found := charstring firstres
      THEN 1 := 1
      ELIF found := charstring lastres
      THEN 1 := top
ELSE
  # seek a match by binary chop #
  REF INT s = LOC INT
  := (top + 1) OVER 2 ;
  i := s ;
  WHILE
    STRING entry = resstrings[1] ;
    NOT (found := charstring=entry)
    AND s > 1
    DO
      s := (s + 1) OVER 2 ;
      IF charstring < entry
      THEN i MINUSAB s
      ELSE i PLUSAB s
      FI
      OD
    FI ;
    IF found
    THEN rword := restable[1]
    FI
  FI ;
  found
END ;

IF pt OR upperstrop AND ul
THEN #3#

PROC (PROC (REF CHAR)BOOL) WORD boldscamer
  = (PROC (REF CHAR)BOOL charbool ) WORD
  BEGIN
    REF STRING boldstring = LOC STRING
    := char ;
    WHILE
      get next character (char) ;
      NOT eol AND charbool(char)
      DO
        boldstring PLUSAB char
      OD ;
    IF
      REF WORD rbw = LOC WORD ;
      matchres(boldstring, rbw)
    THEN
      rbw
    ELSE
      (boldstring, boldtagproc)
    FI
  END ;
IF upperstrop THEN
  IF ul OR pul THEN
    boldscanner ((REF CHAR c)BOOL : uletter(c) OR digit(c))
  ELSE
    point followed by lower case#
    boldscanner ((REF CHAR c)BOOL : sletter(c) OR digit(c))
  FI
ELSE
  boldscanner ((REF CHAR c)BOOL : letter(c) OR digit(c))
FI
ELSE #3#
  # restrop and word does not begin with a point #
  REF BOOL tag held = LOC BOOL := FALSE,
  resposs = LOC BOOL := TRUE,
  resfound = LOC BOOL := FALSE;
  REF STRING tagstring = LOC STRING := emptystring,
  taggle = LOC STRING ;
  REF WORD rbw = LOC WORD ;
  WHILE
taggle := char ;
  WHILE get next character (char) ; NOT eol
    AND (letter(char) OR digit(char))
    DO
taggle PLUSAB char
    OD ;
IF NOT eol AND char=underscore THEN
resposs := FALSE ;
get next character (char)
ELSE
# an apparent taggle may be a reserved bold word if it is bounded by disjunctors and not adjacent to an underscore #
IF resposs THEN resfound :=
matchres(taggle, rbw)
FI
FI ;

break in tag (VOID: resposs := TRUE) ;
# if there are typographical display features then resposs is reset ready for the next apparent taggle #

NOT resfound AND
(BOOL l = letter(char) ;
resposs := resposs AND l ;
l OR digit(char) )
# a taggle may start with a letter or a digit, but every reserved bold word starts with a letter #
DO
tag held := TRUE ;
tagstring PLUSAB taggle
OD ;

# the input may contain a tag followed by an object recognized firstly as an apparent taggle and secondly as a reserved bold word; i.e. two words may be recognized in one activation of the recognizer; alternatively, the first apparent taggle may or may not be a reserved bold word #

IF tag held THEN
IF resfound THEN
word held := TRUE ;
held word := rbw
FI ;
(tagstring, tagproc)
ELSE
rbw
FI
FI #3#
FI #2#
# finished with tags and bold words #
ELIF #1#
dgt OR pdgt
THEN #1#

# an INTREAL-denotation or a bits-denotation
(or a digit-symbol in a priority-definition) #

REF STRING denstring := LOC STRING :=
IF pdgt THEN "0." ELSE emptystring FI + char ;

PROC VOID get digits
= VOID : WHILE get next character (char) ;
NOT eol AND digit(char)
DO denstring PLUSAB char OD ;

PROC BOOL aletterproc
= BOOL :
IF eol
THEN FALSE
ELIF upperstrop
THEN sletter(char)
ELSE letter(char)
FI ;

get digits ;

BOOL aletter = aletterproc ;

IF #2#
dgt AND aletter AND char = "R"
THEN #2#

# a bits-denotation #

denstring PLUSAB "R" ;
REF BOOL radixright := LOC BOOL := TRUE ,
digits = LOC BOOL := FALSE ;

[ : ]CHAR bitsdigits =
IF denstring = "2R" THEN "01"
ELIF denstring = "4R" THEN "0123"
ELIF denstring = "8R" THEN "01234567"
ELIF denstring = "16R"
THEN "0123456789abcdef"
+ IF upperstrop
THEN emptystring
ELSE "ABCDEF"
FI
ELSE radixright := FALSE ;
SKIP
FI ;
IF radixright
THEN
WHILE get next character (char) ; NOT eol AND char in string (char, LOC INT, bitsdigits) DO digits := TRUE ; denstring PLUSAB (sletter(char) ; char) # changes any lower case letters to upper case OD ; IF digits THEN ( denstring, bitsdenproc ) ELSE emit a fault message (no digits in bits-denotation) C ; ( denstring, badbitsdenproc ) FI ELSE C emit a fault message (wrong radix in supposed bits-denotation) C ; WHILE # may maul the next word # get next character (char) ; NOT eol AND ( IF upperstrop THEN sletter(char) ELSE letter(char) FI OR digit(char) ) DO denstring PLUSAB char OD ; ( denstring, badbitsdenproc ) FI
ELIF #2#

BOOL intpoint = dgt AND NOT eol AND char = "." ;

BOOL intandfracpart = IF intpoint
THEN
    get next character(char) ;
    intpointletter := letter(char) ;
    NOT intpointletter
ELSE
    FALSE
    FI,

intandexpart = dgt AND aletter
    AND char = "E" ;

dgt AND NOT( intandfracpart OR intandexpart )

THEN #2#

# an integer-denotation or a digit-symbol #

( denstring, intdenproc )

ELSE #2#

# a real-denotation #

REF BOOL fracright = LOC BOOL ;

BOOL expart = IF pdgt
THEN fracright := TRUE ;
    aletter AND char = "E"
    ELIF intandfracpart
THEN IF fracright := digit(char)
THEN
denstring PLUSAB "." + char ;
    get digits ;
    aletterproc AND char = "E"
    ELSE
    FALSE
    FI

ELSE #intandexpart#
denstring PLUSAB ".0" ;
    fracright := TRUE
    FI ;

IF #3#
expart
THEN #3#
denstring PLUSAB "E" ;
    get next character (char) ;
    IF char = "+" OR char = "-"
    THEN denstring PLUSAB char ;
        get next character (char)
    ELSE denstring PLUSAB "+"
    FI ;
    get digits
FI #3# ;
IF

fracright AND digit(denstring[UPB denstring])

THEN

( denstring, realdenproc )
# integral-part and fractional-part
of denstring will contain
at least the digit 0 #

ELSE

C emit a fault message
   (ill formed real-denotation) C ;

( denstring, badrealdenproc )

FI

FI  #2#

ELIF  #1#

   char = quote

THEN  #1#

   # a character- or string-denotation #

   PROC VOID eol in string
      = VOID : IF eol
         THEN C emit a warning # see HR C4 #
            (string-denotation
             broken by end of line)  C ;
            eol := FALSE
         FI ;

   REF STRING denstring = LOC STRING := emptystring ;
WHILE

get next character (char);  
eol in string;

IF char = apostrophe  
THEN

get next character (char);  
eol in string;

IF char ≠ apostrophe  
THEN  
C # see HR A3.1 # a routine to deal  
with the situation where a single  
apostrophe in a string-denotation  
is used as an escape character,  
otherwise a fault condition  
C

# (two apostrophes form  
the apostrophe-image) #

FI ;  
TRUE

ELIF char = quote  
THEN

get next character (char);  
IF NOT eol AND char = quote  
THEN # quote-image #

TRUE

ELSE WHILE char = space  
DO get next character (char) OD ;  
eol := FALSE ;

IF char = quote  
THEN # string-break, see HR 3.1 #

get next character (char) ;  
TRUE

ELSE # end of string #

FALSE

FI

ELSE

FI

FI

DO
denstring PLUSAB char
OD ;

CASE 1 + UPB denstring

IN ( emptystring, estringdenproc )

, ( denstring, chardenproc )

OUT ( denstring, stringdenproc )

ESAC
ELIF #1#

REF INT dyadnum = LOC INT;
char in string ( char, dyadnum, "%+-<=>/*" )

THEN #1#

# DOP-BECOMESETY-symbol (operator)
and/or is-defined-as-symbol #

PROC (WORDPARAMS) VOID opproc = IF dyadnum <= 3
THEN taoproc # operator could be monadic #
ELSE tadproc # operator must be dyadic #
FI;

REF STRING opstring := char;
get next character (char);

BOOL colon2 = char = ":", equals2 = char = "=";

IF #2#

elo OR 
NOT (colon2 OR char in string(char, LOC INT, "(<>/*"))

THEN #2#

# one character only e.g. "%" or "=" #

(opstring, IF opstring = "=" THEN equalsproc ELSE opproc FI)

ELIF #2#

PROC WORD colonequals = WORD : IF

opstring PLUSAB ":";
get next character (char);
elo OR char ≠ "="

THEN
C emit a fault message
(ill formed operator) C ;
(opstring, badtaoproc)
ELSE

opstring PLUSAB ":";
get next character (char);
(opstring, opproc)
FI;

colon2
THEN  #2#
    colonequals # e.g. "%:=" #
ELSE  #2#
    # second character not ":" #
    opstring PLUSAB char ;
    get next character (char) ;
    BOOL colon3 = char = ":" , equals3 = char = ":=" ;
    IF  #3#
        eol OR NOT(colon3 OR equals3)
    THEN  #3#
        IF 
            equals2  # n.b. second character #
        THEN
            # have we a DYAD-cum-equals-symbol e.g. ":="
            or
            DYAD-symbol, is-defined-as-symbol ?
            Assume the second, think again when the context is determined #
            word held := TRUE ;
            held word := ( ":=" , equalsproc ) ;
            ( opstring[1] , opproc )
        ELSE
            ( opstring , opproc ) # e.g. "%<" #
        FI
        ELIF  #3#
        equals2 AND colon3  # e.g. ":=" #
    THEN  #3#
        opstring PLUSAB ":" ;
        get next character (char) ;
        IF 
            char = ":="
        THEN  # Assume DYAD-cum-assigns-to-symbol, is-defined-as-symbol #
            word held := TRUE ;
            held word := ( ":=" , equalsproc )
        FI ;
        ( opstring , opproc )
        ELIF  #3#
        colon3
    THEN  #3#
        colonequals # e.g. ":=" #
ELSE #3#
  opstring PLUSAB "=" ;
  get next character (char) ;
  IF eol OR char ≠ ";"
  THEN
    # DYAD-cum-NOMAD-symbol, is-defined-as-symbol, e.g. ";<", "=" #
    word held := TRUE ;
    held word := ( "=" , equalsproc ) ;
    ( opstring[1:2] , opproc )
  ELSE
    # DYAD-cum-NOMAD-cum-assigns-to-symbol, e.g. ";<=" #
    opstring PLUSAB ";" ;
    get next character (char) ;
    ( opstring , opproc )
  FI
FI #3#
FI #2#
ELIF #1#
  char = ";:
THEN #1#
  get next character (char) ;
  BOOL eq = char = "=" , slash = char = "/" ;
  IF #2#
    eol OR NOT(eq OR slash)
  THEN #2#
    ( ";:" , colonproc )
  ELIF #2#
    eq
  THEN #2#
    get next character (char) ;
    IF eol OR char ≠ ";:
      THEN
        ( ";:=" , becomesproc )
      ELSE
        get next character (char) ;
        ( ";:=" , isproc )
      FI
ELSE  #2#
    get next character (char) ;
    IF
      eol OR char ≠ ":"  THEN
        C emit a fault message
        (ill formed isnt-symbol) C ;
        (":=/", badisntproc )
    ELSE
      get next character (char) ;
      IF
        eol OR char ≠ ":"  THEN
          C emit a fault message
          (ill formed isnt-symbol) C ;
          (":=/", badisntproc )
        ELSE
          get next character (char) ;
          (":=./":", isntproc )
    FI
  FI  #2#
ELIF  #1#
  char = ":|
  THEN  #1#
    get next character (char) ;
    IF
      eol OR char ≠ ":"  THEN
        (":|", briefthinlineelseoutproc )
      ELSE
        get next character (char) ;
        (":|":", briefelifouseproc )
    FI
ELSE #1#

    C emit a fault message
    (impermissible character) C ;
    CHAR c = char ;
    get next character (char) ;
    ( c , badcharproc )

FI #1#

FI

COMMENT end of PROC get word COMMENT

COMMENT end of token-recognizer algorithm COMMENT
Appendix : Reserved Bold Words

(The algorithm assumes a well-behaved letter collating sequence)

AT, BEGIN, BITS, BOOL, BY, BYTES, CASE, CHANNEL, CHAR, CO,
COMMENT, COMPL, DO, ELIF, ELSE, EMPTY, END, ESAC, EXIT, FALSE,
FI, FILE, FLEX, FOR, FORMAT, FROM, GO, GOTO, HEAP, IF, IN, INT,
IS, ISNT, LOC, LONG, MODE, NIL, OD, OF, OP, OUSE, OUT, PAR, PR,
PRAGMAT, PRIo, PROC, REAL, REF, SEMA, SHORT, SKIP, STRING,
STRUCT, THEN, TO, TRUE, UNION, VOID, WHILE

(Total : 61)

Epilogue

The author will be pleased to hear from anyone who has queries
or finds mistakes, and will undertake to inform the Algol
Bulletin and any individual correspondents of necessary
amendments. Enquiries about the analogous Algol 68R program
are also invited.

Please write to:

Mr. R. Bell,
Department of Computer Science,
Teesside Polytechnic,
Borough Road,
Middlesbrough,
Cleveland,
TS1 3BA,
England.