ALGORITHM 268

ALGOL 60 REFERENCE LANGUAGE EDITOR [R2]

W. M. McKeeman* (Recd. 9 Dec. 1964, 23 Feb. 1965 and 17 May 1965)

Computer Science Department, Stanford University, Stanford, California

* Supported in part by the Office of Naval Research under Contract Nonr 225(37), NR 044-211.

The author expresses his thanks to the referee for several valuable suggestions.

```
procedure Algoledit(characterset, linelimit);
string characterset;
integer linelimit;
comment If this procedure is presented an ALGOL 60 program
or procedure in the form of a sequence of basic symbols, it will
transmit to the output medium a copy of the text with indenta-
tions between each begin-end pair and some rearrangement of the
blank spaces within the text. This procedure is an example of its
own output. It is used to edit ALGOL 60 text that is difficult to read
because, for example, the ALGOL has been transcribed from
printed documents, or written by inexperienced programmers, or
stored in compressed form (i.e., with all redundant blank spaces
removed). The integer "—1" will represent the nonbasic symbol
"carriage return", "—2" will represent an end-of-file mark, other
symbols will have the integer value corresponding to their position
in the parametric string "characterset". The string must contain
exactly the 116 basic symbols of ALGOL 60. The parameter "line-
limit" sets an upper bound on the number of basic symbols that
the user wishes to appear on a line of output. The identifiers
"lsq" and "rsq" will be used in place of strings of length one whose
only elements are " ' " and " ' ", respectively;
begin integer array spacesbefore, spacesafter[1 : 116],
    buffer[1 : linelimit];
    integer tabstop, symbol, i, symbolcount, level;
    Boolean newline;
    integer procedure val(s);
    string s;
    comment The value of this procedure is the integer
    corresponding to the position in the string "characterset"
    of the symbol in the string "s". The body of the
    procedure must be expressed in code;
    procedure get(symbol);
    integer symbol;
    begin insymbol(2, characterset, symbol);
        if symbol = -2 then go to eof
    end get;
    procedure send(symbol);
    integer symbol;
    begin comment "send" must not break identifiers
        across lines or insert spurious characters into
        strings;
        integer i, u, v;
        if symbol = -1 ∨ symbolcount ≥ linelimit
        then
        begin v := tabstop;
            if newline then go to E;
            if level ≠ 0 then
            begin comment Inside a string;
                for i := 1 step 1 until
                symbolcount do outsymbol(1,
                characterset, buffer[i]);
                outsymbol(1, characterset, -1);
                v := 0
            end else
            begin u := symbolcount;
                newline := true;
                if symbol = -1 then go to D;
                comment Find a convenient place to
                break the line;
                for u := symbolcount - 1 step -
                1 until 1 do if buffer[u + 1] =
                val('␣') ∨ buffer[u] = val(rsq) then
                go to D;
                u := symbolcount;
                comment Send the line;
                D : for i := 1 step 1 until u do
                outsymbol(1, characterset, buffer[i]);
                outsymbol(1, characterset, -1);
                comment Find a non-blank character
                to start the next line;
                for i := u + 1 step 1 until
                symbolcount do if buffer[i] ≠ val('␣')
                then go to F;
                go to G;
                comment Move a new line to the
                head of the buffer area;
                F : for i := i step 1 until
                symbolcount do
                begin v := v + 1;
                    newline := false;
                    buffer[v] := buffer[i]
                end;
                comment Insert blanks for tab stops;
                G : for i := 1 step 1 until
                tabstop do buffer[i] := val('␣')
            end;
            E : symbolcount := v
        end;
        comment Now we can put the new symbol in the
        buffer array;
        if symbol ≠ -1 ∧ ¬ (newline ∧ symbol
        = val('␣')) then
        begin symbolcount := symbolcount + 1;
            newline := false;
            buffer[symbolcount] := symbol
        end
    end send;
    for symbol := 1 step 1 until 116 do
    spacesbefore[symbol] := spacesafter[symbol] := 0;
    for symbol := val('+'), val('-'), val('¬'), val(':'),
    val(':='), val('<'), val('≤'), val('='), val('≠'),
    val('≥'), val('>') do spacesbefore[symbol] :=
    spacesafter[symbol] := 1;
```

```
    for symbol := val('∧'), val('∨'), val('⊃'), val('≡'),
    val('then'), val('else'), val('step'), val('until'),
    val('while'), val('do') do spacesbefore[symbol] :=
    spacesafter[symbol] := 2;
    for symbol := val('go to'), val('begin'), val('if'),
    val('for'), val('procedure'), val('value'), val('own'),
    val('real'), val('Boolean'), val('integer'), val('array'),
    val('switch'), val('label'), val('string'), val(',') do
    spacesafter[symbol] := 2;
    level := symbolcount := tabstop := 0;
    newline := true;
    nextsymbol : deblank : get(symbol);
    scanned : if symbol = val('␣') ∨ symbol = - 1
    then go to deblank;
    if symbol = val('begin') then send( - 1) else
    if symbol = val('end') then
    begin tabstop := tabstop - 5;
        send( - 1)
    end;
    for i := 1 step 1 until spacesbefore[symbol] do
    send(val('␣'));
    send(symbol);
    for i := 1 step 1 until spacesafter[symbol] do
    send(val('␣'));
    if symbol = val('comment') then
    begin comment Pass comments on unchanged;
        for i := 1 while symbol ≠ val(';') do
        begin get(symbol);
            send(symbol)
        end
    end else if symbol = val('end') then
    begin comment "end" comments;
        for i := 1 while symbol ≠ val(';') do
        begin get(symbol);
            if symbol = val('else') ∨ symbol =
            val('end') then go to scanned;
            send(symbol)
        end
    end else if symbol = val(lsq) then
    begin comment Pass strings on unchanged;
        level := 1;
        for i := 1 while level ≠ 0 do
        begin get(symbol);
            send(symbol);
            if symbol = val(lsq) then level :=
            + 1 else if symbol = val(rsq)
            then level := level - 1
        end
    end;
    if symbol = val('begin') then tabstop := tabstop + 5
    else if symbol = val(';') then send( - 1);
    go to nextsymbol;
    eof : send( - 1);
    outsymbol(1, characterset, - 2)
end Algoledit
```

REMARK ON ALGORITHM 268 [R2]

ALGOL 60 REFERENCE LANGUAGE EDITOR
[W. M. McKeeman, *Comm. ACM 8* (Nov. 1965), 667]
G. Sauer (Recd. 23 Dec. 1968)
Institut für Theoretische Physik der Justus-Liebig-Universität, 63 Giessen, West Germany

KEY WORDS AND PHRASES: symbol manipulation
CR CATEGORIES: 4.49

In the **procedure** *send*, replace the line

```
1 until 1 do if buffer[u+1] =
```

with the line

```
1 until tabstop do if buffer[u+1] =
```
(1)

The published version fails to clear the buffer when a line to be printed contains no blanks and *tabstop* > 0, causing an array bounds violation. Knowing *buffer*[*tabstop*+1] never to contain a blank character, the search for blanks may be stopped at *u* = *tabstop* + 1.

---

(1) The author is indebted to the referee for suggesting this brief form.