NATIONAL PHYSICAL LABORATORY

CENTRAL COMPUTER UNIT

SOAP - SIMPLIFY OBSCURE ALGOL PROGRAMS

by

R.S. Scowen, A.L. Hillman and M. Shimell

A Station of the Ministry of Technology
Extracts from this report may be reproduced provided the source is acknowledged.
SOAP - Simplify Obscure Algol Programs

by

R.S. Scowen, A.L. Hillman and M. Shimell

Abstract

SOAP is a KDF9 Algol program which reads an Algol program as data, cleans it up, and outputs it in a form which displays its structure.
Contents

1. Introduction
2. How to use SOAP
3. Program Failures
4. What SOAP does
5. More properties of SOAP
6. Method
7. Size and Speed
8. Acknowledgements
9. Listing of SOAP
1. Introduction

SOAP is a program which acts as an editor. It reads an Algol 60 program as data, cleans it up, and outputs it in a form which displays its structure. SOAP is written in Algol 60.

There is no standard for specifying the best layout for published Algol programs. The Algorithms Sections of various journals, e.g. Comm. ACM, Comp. J., Numer. Math., BIT, generally conform to their own standard but no two of them are the same. SOAP edits programs differently from all these journals, the method it uses makes it easier for the reader to find the extent of any statement or comment and easier to find the declaration of any identifier.

The general rule which is adopted in the editor is: "if a structure extends over more than one line, subsequent lines are indented." Comments, declarations, for statements, and assignment statements all satisfy the rule. Conditional statements are one exception: the structure is traced as follows: start at 'if', evaluate the boolean expression, if the value is true then first trace the indented statement and then jump to the first basic symbol under 'if' which is not 'else'; if the value is false jump to the next symbol under 'if' and so on. Labels are another exception; they stick out a little so that it is easy to find them (note that they are nearly always indented less than the 'goto'; the only exceptions are jumps into the middle of conditional or compound statements). The declarations and statements of a block are not indented if the 'begin' and 'end' have themselves been indented (i.e. after 'then' or 'do'). This suppresses unnecessary indentation.

Indentation is achieved by outputting zero or more tabulation characters at the beginning of each line. Henceforth normal KDF9 usage is adopted and a tabulation character is called a tab.

Experience of writing Algol programs has been the main factor in determining the properties of SOAP. Other causes have been (1) what could be programmed fairly simply, (2) modifications shown to be desirable after a few test runs of SOAP, (3) discussions with other programmers.

2. How to use SOAP

SOAP is an Algol 60 program and its call tape on KDF9 is:

```
K
JX8890---KIP;
PROGRAM 3200;
IN 8;
OUT 8; L.
```

The Algol program which is to be edited can be read either from 8-hole paper tape or from the 'Prompt' system on the disc. The output can be sent to either an 8-hole paper tape punch or a line printer.

SOAP starts by reading a data tape from an 8-hole paper tape reader. The syntax of this data tape is as follows:

```
<data tape> ::= <output device number> <number of programs>
              <list of programs>  
          <output device number> ::= 10; / 30;
          <number of programs> ::= <integer> ;
```
<list of programs> ::= <length of output line> <program> /
   <length of output line> <program> <list of programs>
   <length of output line> ::= <integer> ;
<program> ::= 20 <Algol program> /
   120; [12<twelve character identifier of Algol program in
   Prompt system > ];

This looks far more complicated than it really is; the following data
tape was used to obtain the listing of SOAP given in Appendix 1:--

   10; 1; 100; 120; [JXES810--APU ]; ✔

Note that minus signs in the program identifier have the usual 'Prompt'
meaning.

SOAP is kept as a compiled program on the NPL magnetic tape DE021E11N.

3. Program Failures

SOAP may fail while it is running. This section explains the possible
causes.

(1) The initial data is incorrect

Most errors in the data tape will cause SOAP to fail and print
a message on the printer or output device. The message is either 'error in
initial data' or 'error in data'.

(2) The Algol program is incorrect

SOAP will edit most Algol programs even incorrect ones, however it
is possible for SOAP to print a message 'fail n' and stop editing. The possible
values of 'n' and the reasons for the failure are given below:--

<table>
<thead>
<tr>
<th>'n'</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>100, 105, 109</td>
<td>The value of 'length of output line' is too small.</td>
</tr>
<tr>
<td>108, 111</td>
<td>A library call is not followed by a comment.</td>
</tr>
<tr>
<td>110</td>
<td>The number of 'begin's is not equal to the number of 'end's.</td>
</tr>
<tr>
<td>107, 112</td>
<td>There is some other error in the Algol program.</td>
</tr>
<tr>
<td>113</td>
<td>One of the lines in a comment is too long.</td>
</tr>
<tr>
<td>Any other number</td>
<td>You have found an error in SOAP, please contact the author so that it may be put right.</td>
</tr>
</tbody>
</table>

(3) Failures while finding the Algol program on the disc.

A message 'FAIL n IN FINDPROG' is output

<table>
<thead>
<tr>
<th>'n'</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Invalid character inside the string on the datatape.</td>
</tr>
<tr>
<td>2</td>
<td>Not twelve characters inside the string.</td>
</tr>
<tr>
<td>3,4</td>
<td>Error in SOAP</td>
</tr>
<tr>
<td>5</td>
<td>Program to be edited is not on the disc.</td>
</tr>
<tr>
<td>10,11</td>
<td>Prompt error.</td>
</tr>
</tbody>
</table>
(4) Failures while reading the Algol program from the disc

A message 'FAIL n IN HEADPROG' is output

' n' Reason
0, 10 Prompt error.
4 More 'begin's than 'end's in the program.

4. What SOAP does

This section is a list of some of the properties of an Algol program
after it has been edited by SOAP.

4.1 One statement per line

All statements occupy at least one line; the only exception is
an unlabelled dummy statement immediately before 'end', e.g.:-

statement;
statement;
statement;
;
statement
end

4.2 Corresponding 'begin'/ 'end' symbols are lined up

Corresponding 'begin's and 'end's are on a line by themselves and
indented to the same extent. The declarations and statements of nested blocks
and compound statements start with one more tab than the 'begin' and 'end'; e.g.:-

begin
  statements;
begin
  declarations;
  statements
end;
  statements
end

4.3 Long statements are tabbed in after the first line

Any statement which is too long to go on one line is indented on
subsequent lines. The line is broken at the last space character outside a
string; e.g.:-

statement;
this statement is too long to go on one line and is
indented on later lines;
statement
4.4 For statements

The controlled statement of a for statement starts on a new line and is indented; e.g.:-

    for i := for list do
    for j := for list do
        statement;

4.5 Conditional statements

In a conditional statement the statement after 'then' starts on a new line with an extra tab. 'else' is put on a new line under 'if'; if the statement after 'else' is a conditional statement it starts immediately after the 'else', otherwise it is put on a new line and indented; e.g.:-

    if conditional expression then
        statement
    else if conditional expression then
        statement
    else
        statement;
    statement

4.6 Conditional expressions

If a conditional expression is not enclosed in brackets it is treated in a similar way to conditional statements; e.g.:-

    a := b :=
    if condition then
        expression
    else
        expression;
    p (if condition then expression else expression)

4.7 Labels on separate lines

A label is put on a separate line with 'half a tab' less than the statement being labelled; that is, it is indented less than the following statement but more than the corresponding 'begin' and 'end'. e.g.:-

    begin
    statement;
    label:
    statement
4.8 Extra dummy line after a procedure declaration

Every procedure declaration is followed by a dummy line; e.g.:--

    procedure p (formal);
    real
    formal;
    statement;
    declaration;

4.9 Comments are copied with extra dummy lines

A comment starting with 'comment' is copied as it was written except that (1) it
has an empty line before and after it, (2) it is moved so that it
lines up with the declarations or statements it follows, (3) tab is replaced by
space; e.g.:--

    statement;
    comment note that there is a dummy line before and after
    this comment, note also that every line after the first
    is indented;
    statement;

4.10 Each declaration is on a separate line

All variables, arrays, switch elements, etc. are declared and specified
on a separate line, e.g.:--

    real
    alpha,
    beta;

    array
    square 1,  
square 2 [1 : n, 1 : n],
    rectangle [0 : m, 0 : n];

5. More properties of SOAP

The properties of SOAP which have been described are those applicable with
programs in pure Algol 60. This section describes facilities which are concerned
with the special features of KDF9 Algol.
5.1 Input from paper tape or disc

The Algol program which is to be edited can be read from 8-hole KDF9 paper tape or from the disc if it is in the Prompt operating system.

5.2 Output to paper tape or line-printer

The Algol program which has been edited can be output either on a paper tape punch or onto a line printer.

5.3 Putting the Algol program back into 'Prompt'

If the Algol program is read from the 'Prompt' system, an 'Establish' message is output at the start of the program so that the edited version can be reinput if desired.

5.4 Gaps on the output tape

If the Algol program is output onto paper tape, a gap of 50 characters is output after every 32 lines.

5.5 Code bodies and library calls

Procedures with code bodies can also be edited. Algol and Usercode library calls are reproduced if they are followed by 'Facsimile' type comments which specify them. These two facilities are not available if the Algol program was read from paper tape.

6. Method

SOAP does its editing by partially analysing the syntax of the Algol program. Inside the editor there are separate procedures to edit different syntactic parts of an Algol program, e.g. block, statement, specifications or declarations, comments, bracketed structures. These procedures call one another in a mutually recursive way, e.g. while editing a program it is possible that at one point: 'block' calls 'statement' calls 'block' calls 'specifications or declarations' calls 'proc declaration' calls 'statement'. The recursive procedures have no parameters called by name, and a compound statement (not a block) as body; they communicate their results using a small number of variables which are global to SOAP. These features simplify a manual rewriting of SOAP into an assembly language, because the recursive procedures can be written as subroutines which stack the return address on entry, and unstack it on exit.

SOAP makes a single pass through the Algol program, it reads one basic symbol at a time using the procedure 'read symbol' and outputs one line at a time using the procedure 'out line'.

The methods and techniques used in SOAP were originally developed for the Babel compiler. (Babel is a high-level computer language and compiler being developed at NPL).

7. Size and Speed

The version of SOAP written to deal with full KDF9 Algol is compiled into about 3000 words of KDF9 code in the Kidsgrove Algol system. SOAP itself is about 2000 words and the rest (mainly input/output routines) is about 1000 words.
SOAP needs runtime space for arrays, variables and stacks of about 1700 words. The largest item is 640 words for an input buffer from the disc.

The KDF9 Kidsgrove Algol version of SOAP edits itself into 1400 lines of Algol and takes 130 seconds.

8. Acknowledgements

The first version of SOAP was written by M. Shimell (University College Oxford) and R.S. Scowen. The program described here is an extended and modified version of the original.

The procedures to find and read an Algol program from the disc were written by A. Hillman.

I am also grateful for discussions with Mrs. M. Price and Dr. B. Wishmann on the desired properties of SOAP.

9. A listing of SOAP

This section is a listing of SOAP which has been produced by applying it to itself. Thus the listing shows both what SOAP does and how it is done.

The line at the top of each page is not part of the output: it has been included in order to show how far each page is indented.

VP
begin

library MUX-5,2001;

comment MUX-5,2001;

library ac912;

comment ac912;

procedure findrow(dv, m);
value
ov;

integer
ov;

integer array
a;

KDR3 3 / 9 / 4 / 5;

V0=0 16 3 5 2 4 21 46 24 60 20 74 33;

V1=0 07 21 35 22 41 26 00 00 00

V2=0 07 21 64 02 00 04 00 00 04

V3 = 0 0 / 3 / 1

SIGN10; [a1]; DIN24; ENL-16; +1; D17;

SHIFT; ADDI; SET6005; +1; J86529;

ENL-32; +1; J86501;

DIF; V3; +1; -V36605;

DIF; ENL-16; +1; SET6039; +1; -V36605;

[a];

J865695; SET6159;

1j; J865299; J1f; J1f; SET12; +27;

UNASP;

TARP; ZINC;

7j; J865955; DOMIN; DURAN; DURAN; SET6295; J6=1;

SET6029; J6=4;

DIF; SET6055; +1; J86529;

DIF; SET6345; +1; J86529;

DIF; SET6295; +1; J86529;

SET6295; +1

8j; DOM; DURAN; DURAN; PERM;

IV; NEW; J863729; J865295;

DOMIN; DURAN; DURAN;

SET6295; NEW; 46=5;

J865295;

V3 = ov; j10; SET6055; J9; XXX;

9j; PLAIN; SET6059; D39;

10j; SET6059; +1; J39;

11j; SET6059; J9;

12j; SET6059;

13j; SET6059; SET6059; GT+1 CAR4;

J865951;

AL/063;
Integer procedure read-next;

1002 V / H / A / H;
20 D V B V D V A V B 20 72 31;
300 V H V 2 20 20 20 20 00 00;
400 \#Mb13 111 \#Mb11;
50 8B7A\#CC 285\#C 129\#CC;

\#Mb11V6; ( Finds algol basic symbols one at a time );
100 V=0D/3/1H;
200 V=0D/VH;
300 V=0D;
400 V=13;
500 V=\#R12;
600 V=\#R11;
700 V=\#R15; V11 = \#Q14;
800 V11 = \#R13;
900 V=13;
100 Spy 13;
999 Spy 13; EXIT 13;
1000 V4; V11 = -V4; EXIT 13; 31; NEWAGE;
11 V2; J152V; J151J13E2; J131;
900 CUTOX30; V23 BUX; J100=2S;
1324 J102=12; J275+; JAN; JN112;
275=2B5;
130 V=3; MON14;
135 M15; M115; M15; M15; M15; M15; M15;
1300 EN2M; M10(2); V13; MIN12;
1636 M16; M12; SET2C; +3 = M18;
500 MON12; J10; -V2;
11 SET 6; =C15; J131J82;
500 DUX; ZER03; SHL=8;
100 SHL+43; SHL-43; J133;
200 J275; SHL43; J132;
300 B015; J107;
800 C013; J012\#C; NEWAE;
100 V2; ZER04; SHL=8; DC15;
100 ZER04; V65;
200 =V23; EXIT 877; J3=1;
300 SET276; J16=1;
121 C13; =M15;
Q15; =V01 C14;
-\#V4; EXIT 81;
144 ZER04; HTT; =V4;
113 ZER04; =V2; ZER04; =V11; ZER04; =V04; EXIT 81; 113 SET275; ZER04; HTT; =V6;
113 ZER04; =V6; CUTOX30; J132;
71 DC13; C13; DUX; RES; =C14;
200 =\#M14; CUTOX30; V6; J105=2; J275=82; J132;
41 ZER03; SHL=8; SHL=8;
100 V65\#CC; 013; 013; SHL=81;
100 Spy 13; \#CC; EXIT 13.
procedure printtitle(n);  
value  
n;  
integer  
n;  
end;  

procedure print(s, n);  
value  
s;  
string  
s;  
integer  
n;  
begin  
write(s);  
write(s, format(' %s', n));  
end;  
end;
procedure algoedit(line-limit, id, od, failure);
    value
      line-limit,
      id,
      od;
    integer
      line-limit,
      id,
      od;
  label
  failure;

comment This procedure edits one Algol 60 program when it is called.
The meaning of the parameters is:
'line limit' - the maximum number of basic symbols to be output on one line
'id' - the input device number
'od' - the output device number
'failure' - the procedure jumps to this label if a failure occurs while
  editing is in process;

begin

comment 'Algol edit' uses arrays for the following purposes:
'n tab' - the elements of this array are used to remember the number
  of tabs which must precede the start of each statement and declaration
  of every block
'disc buffer' - this array is used as a buffer if the Algol program is read from the disc
'buffer' - this array is used as a buffer to hold the symbols which will
  be output on the next line
'a' - this array is used as a temporary store when a line must be split
  because it is too long
'and, 'up' - these arrays are used to specify the number of spaces which
  are to be output before and after each different basic symbol

integer array
  ntab[1:25],
  buffer,
  s[0:150],
  discbuffer[0:if id = 10 then 640 else 1],
  spn,
  up[0:255];
comment  At any point in the Alcol program being edited, these integer variables have the following values:

'b str', 'a str' - the number of lexems and sym's that have occurred so far
'bs' - the current basic symbol
'depth' - the current nested block depth, i.e. 'b str' - 'a str'
'i' - this is used as the controlled variable in a for statement
'lo' - the next symbol to be output will be put in 'buffer[lo]'
'line number' - the value of this variable is the number of lines which are to be output before the next gap
'start of string' - this variable is used in the procedures 'read string' and 'copy string'. It's value indicates the start of the current string in the output buffer and is needed if the string is too long to be put on one line
'saba' - the number of tab symbols which must start the next line to be output
'tabspace' - the number of space symbols equivalent to one tab symbol.

integer

bstr,  
bs,  
depth,  
cstr,  
i,  
lo,  
linenumber,  
startofstring,  
tabs,  
tabspace;
comment Each one of these variables represents an Algol basic symbol and has an appropriate constant value:

integer
capitals,
and,
array,
because,
begin,
boolean,
colon,
comma,
comment,
divide,
do,
dot,
dotless,ends,
equal,
equal.
eq,
false,
for,
goto,
greaterthan,
greatequal,
if,
imp,
intdiv,
integer,
label,
leftbracket,
leftbracket,
leftparen,
lessthan,
lessorequal,
minus,
multiply,
newline,
nine,
not,
otequal,
or,
over,
plus,
procedure,
real,
rightbracket,
rightbracket,
semicolon,
space,
step,
string,
substring,
switch,
tab,
then,
true,
until,
value,
while,
zero,
nullval
comment: Each one of these variables represents a KDPS pseudo basic symbol and has the appropriate constant value;

integer
   algol,
   endmessage,
   exit,
   kdry,
   library,
   segment,
   tandummy;

comment A list of the procedures in algol edit.
read symbol
out line ( integer value ad, integer array buffer, integer value le )
next line
comment statement
out i ( integer value char )
out o ( integer value char )
clear full line ( integer value symbol )
specifications or declarations ( boolean value declarations )
proc declaration
fall ( integer value n )
expression ( integer value symbol )
for variable and list
if clause
statement
possible label( boolean value inserting a dummy statement is possible)
copy string
read string
copy square brackets
copy round brackets
block ( boolean value inner )
call library
code body
;
procedure reasymbol;

comment This procedure assigns the next basic symbol of the Algol program being edited to the global variable 'bs'. It is a machine dependent procedure;

begin
if bs = endmessage then
   fail(14);
bs :=
if td = 120 then
   readprog
else
   inbasymbol[td];
end readsymbol;

procedure outline(od, buffer, lci);

value od,
lci;

integer od,
lci;
integer array buffer;

comment This is the basic output routine which prints the next line of the program on device od. The symbols of the output line are stored in elements 1 to 20 of the array buffer. outline is a machine dependent procedure;
procedure nos;

argument
'nos' assigns the next non-printing symbol of the Algol program
being edited to the variable 'be';

begin
label7 ;;
readsymbol;
if be = space or be = newline or be = tab then
    goto label7;
end nos;

purpose procedure letterordigit:
letterordigit := (be = capitals and be < smalls) or (be = zero and be < nine);

procedure identifierorlabel;

comment
This procedure copies successive symbols of the Algol program which
form either an identifier or label;

begin
    if not letterordigit then
        fail(true);
    label8 ;;
    out(be);
    nos;
    if letterordigit then
        goto label8;
    end identifierorlabel;

procedure scan(symbol1, symbol2);

value
symbol1, symbol2;

integer
symbol1, symbol2;

comment
'scan' copies successive non-printing symbols and bracketed elements
of the Algol program. It inserts spaces where appropriate and stops
when the current basic symbol is either 'symbol1' or 'symbol2';

begin
label2 ;;
    if be = lbracket then
        begin
        copyroundbrackets;
        goto label2
        end;
    if be = lbracket then
        begin
        copysquarebrackets;
        goto label2
        end;
    out(be);
    if be = symbol1 and be = symbol2 then
        begin
        nos;
        goto label2;
        end
end scan;
procedure writeln;
  comment "next line" outputs the next line of the edited Algol program and
  stores in "buffer" the tab symbols at the beginning of the next line;
begin
  integer
    1,
    2;
  if line_number = 0 then
    begin
      if od = 10 then
        begin
          outline(od, buffer, le - 1);
          gap(od, 50);
        end
      else
        begin
          buffer[16] := newline;
          outline(od, buffer, le);
        end;
      line_number := 31;
    end
  else
    begin
      line_number := line_number - 1;
      buffer[16] := newline;
      outline(od, buffer, le);
    end;
  le := tab_stop * tabs;
  for i := 1 step tab_stop until le do
    begin
      buffer[i] := tab;
      for j := 1 + 1 step 1 until i + tab_stop - 1 do
        buffer[j] := tabsym;
      end for i;
    le := le + 1;
  end writeln;
procedure commentstatement;

  comment "comment statement" edits a comment from the basic symbol `comment' up to the semicolon

begin
  newline;
  tabs := tabs + 1;
  out(tabs);
  label i:
  if tabs > 150 then
    fail(111);
  readsymbol;
  if tabs / semicolon then
    begin
      if tabs = newline then
        newline
      else if tabs = tab then
        outl(espaco)
      else
        outl(tabs);
      goto label i;
      end;
    outl(semicolon);
    tabs := tabs - 1;
  newline;
  newline;
  tabs;
end commentstatement;

procedure outl(char);

value
char;

integer
char;

comment "out l" inserts the symbol 'char' into the output buffer and increases the counter 'lc' by one

begin
  buffer[lc] := char;
  lc := lc + 1
end outl;
procedure out(char); value char; integer char; begin 'out' inserts the symbol 'char' into the output buffer. If necessary 'out'
also puts a space before and/or after 'char'. 'out' also checks to see
if the buffer is full, if so it is emptied;

label
if buffer[lc-1] = subscription then
  begin, if char = plus or char = minus then
    begin
      out(char); msg;
      out(ba);
      goto label17
    end
  end
if (char # 0 and buffer[lc-1] # space) then
  out(char);
if (char = commas or char = semicolon) and buffer[lc-1] = space then
  lo := lo - 1;
if char # 0 then
  out(spacer);
label17:
  if lo > linelimit + 3 then
    fail(idc);
  if lo = linelimit then
    clearfullline(msg);
end out character;
procedure clearFullLine(symbol);
  var
    symbol;
    Integer
    n,
    k;
  begin
    j := lo - 1;
    for k := 1 step 1 until lineLimit = tamanho * (tabs - 1) + 1 do
      begin
        if buffer[j] = symbol then
          goto label1;
        s[d] := buffer[j];
        j := j + 1;
      end;
    fail(100);
  label:
    lo := j;
    if symbol = sentecion then
      begin
        clear(sentecion);
        tabs := tabs + 1;
        nextLine;
        lo := lo - 3;
        for i := 1 step 1 until k - 1 do
          buffer[lo + i] := s[k-i];
        lo := lo + k;
        tabs := tabs - 1;
      end clearFullLine;
procedure specifications or declarations();
value;
begin:
    declarations;
end;
procedure
    declarations;
comment
    if the parameter of 'specifications or declaration' is false,
    then this procedure edits the value and specification part of a
    procedure declaration. If the parameter is true, then the procedure
    edits a list of declarations separated by semicolons;
begin
    label1;
if he = procedure and declarations then
    begin
        proceduredeclaration;
        goto label1;
    end;
else if he = switch and declarations then
    begin
        scan( becomes, becomes);
        next;
    end;
else if he = library then
    begin
        calllibrary;
        goto label1;
    end;
else if he = comment then
    begin
        convertstatement;
        goto label1;
    end;
else if he = real or he = integer or he = boolean or he = array or he = switch or he = label or he = string or he = case or he = value or he = procedure then
    begin
        out(he);
        next;
        goto label1;
    end;
if lc /= tabspace = tabs + 1 then
    begin
        tabs := tabs + 1;
        nextline;
    label1b :
    scan( comm, semicolon);
    if he = semicolon then
        tabs := tabs - 1;
        nextline;
    if he = comma then
    begin
        next;
        goto label1b;
    end;
    next;
    goto label1;
end;
end specifications or declarations;
procedure procdeclaration;

comment 'proc declaration' edits a procedure declaration. The call of the
statement must be extended if it is necessary to take account of
procedures with code bodies;

begin
  scan(semicolon, semicolon);
  tabs := tabs + 1;
  newline;
  ndt;
  specificationsprocdeclarations(false);
  if bs = segment then
    scan(semicolon, semicolon)
  else
    begin
      if bs = xtry then
        codebody
      else
        statement;
      if bs = semicolon then
        out(bs)
      else
        fail(107);
    end;
  tabs := tabs[depth];
  newline;
  newline;
  ndt;
end procdeclaration;
procedure fail(n);
    value
        n;
    integer
        n;
end;

'fail' outputs the current line and a brief failure message.
It then looks for the end of the program and exits to the label
'failure'.

The procedures in which the various failure numbers are generated are:
100 clear full line
101 read string
102 copy square brackets
103 copy round brackets
104 block
105 identifier or label
106 out
107 proc declaration
108 call library
109 read string
110 block
111 code body
112 node body
113 comment statement
114 read symbol
115 for variable and list
117 if clause

begin

le :=
    if le > linelimit then
        linelimit
    else
        le;
nextline;
print ( [ fail , n] ;
label6 ;
readsymbol;
if ha = begin then
    botr := botr + 1
else if ha = end then
    entr := entr + 1;
if botr = entr then
    goto failure
else
    goto label6;
end fail;
procedure expression(symbol);
    value
    symbol;
    Integer
    symbol;

count:
    This procedure edits a conditional or simple expression. It stops when the current basic symbol is and or 'comma' or if or
    while or 'symbol';

begin
if be = if then
    begin
    tabs := tabs + 1;
    newline;
    label0 ;
    elseif;
    expression(else);
    tabs := tabs - 1;
    newline;
    out(else);
    nbs;
    if be = if then
        begin
        outl(special);
        goto label9
        end;
    tabs := tabs + 1;
    newline;
    expression(symbol);
    tabs := tabs - 2
    end
else
    begin
    label0 ;
    if be = lbracket then
        copyroundbrackets
    else if be = lbracket then
        copysquarebrackets;
    if be = end or be = comma or be = step or be = while or be = symbol then

    else
        begin
        out(bs);
        nbs;
        goto label9;
        end;
    end
    goto expression;
procedure forvariableandlist;

    comment: for variable and list edits the first part of a for statement, from for up to and including do ;

    begin
    if be < for then
        fail(114);  
        tabs := tabs + 1;
        scan(becomes, becomes);
        label1 ;
    nbs ;
    scan(comma, do);  
    newline;
    if be = comma then
        goto label0 ;
    nbs
    end for variable and list;

procedure ifclause ;

    comment: This procedure edits an if clause ;

    begin
    if be < if then
        fail(117);  
        out(be);  
        nbs ;
        expression(then);
        out(be);  
        tabs := tabs + 1;
        newline;
    nbs
    end if clause;
procedure statement;

current  statement' quotes any unlabelled statement;

begin
label1 :;

if letterdigit then
    begin
    possiblelabel(false);
goto label1;
    end;

if be = if then
    begin
    iftrue;
goto label1;
    end;

if be = for then
    begin
    forvariableandlist;
goto label1;
    end;

if be = begin then
    block(false);
labels2 :;

if be = ibracket then
    copyroundbrackets
else if be = lbracket then
    copysquarebrackets
else if be = if then
    expression(semicolons)
else if be = else then
    begin
    tabs := tabs + 1;
    nextline;
    out("else");
    no;
    if be = if then
        out(space)
    else
        begin
    tabs := tabs + 1;
    nextline;
    end;
    goto label1;
    end;

if be != semicolons and be != end then
    begin
    out(be);
    no;
goto label2;
    end;
end statements;
procedure possible(label(insertingdumystatementimpossible); value
insertingdumystatementimpossible;
boolean
insertingdumystatementimpossible;

comment 'possible label' is called at the beginning of a statement if the statement
starts with a letter or digit. 'possible label' looks to see if this
identifier or integer is followed by a colon; if it is then the
statement is labelled, the label is shifted half a tab to
the left and put on a line by itself;

begin
integer
1;
d1;
identifierrolabel;
if te = colon then
begin
if tabe = 0 then
begin
d1 := (tabspace + 1) * 2;
l0 := l0 + d1;
for i := tabspace * (size + 1) - d1 step 1 until l0 - 1 do
buffer[i] := buffer[i + d1];
for i := (tabspace * (laks - 1) + 1) step 1 until tabspace * tabs - d1 do
buffer[i] := space;
end;
cut(colen);
hsh;
if be / semicolon then
begin
if insertingdumystatementimpossible then
begin
cut[semicolon];
metline
end
end;
end
end possible label;
procedure copystring;

    comment The two procedures 'copy string' and 'read string' edit a string
    (including nested strings). Editing characters are also copied
    except any tab symbols which occur immediately after a new line symbol;

    begin
    if buffer[lo-1] # space then
       out(space);
    startofstring := lo;
    readstring;
    end copy string;

procedure readstring;

    comment The two procedures 'copy string' and 'read string' edit a string
    (including nested strings). Editing characters are also copied
    except any tab symbols which occur immediately after a new line symbol;

    begin

    integer i;
    if bs / listbracket then
       fail(101);
    label1:
    out((bs));
    if lo > linelimit then
       begin
       lo := startofstring;
       tabs := tabs + 1;
       newline;
       tabs := tabs - 1;
       if lo > startofstring then
          fail(109);
       startofstring := lo;
       for i := startofstring step 1 until linelimit do
          out((buffer[i]));
       end;
    readstring;
    if bs = listbracket then
       begin
    readstring;
    goto label1;
    end
    else if bs = newline then
       begin
    tabs := tabs + 1;
    newline;
    tabs := tabs - 1;
    startofstring := lo;
    new;
    goto label1;
    end
    else if bs / listbracket then
       goto label1;
    end read string;
procedure copysquarebrackets;

comment 'copy square brackets' edits a balanced syntactic structure enclosed in square brackets;

begin
if ba / lbracket then
  ' fall(102);
label5 :;
out(in);
mb;
if ba = rbracket then
begin
 copysquarebrackets;
goto label5
end;
if ba = lbracket then
begin
 copyroundbrackets;
goto label7
end;
if ba / rbracket then
 goto label5;
end copy square brackets;

procedure copyroundbrackets;

comment 'copy round brackets' edits a balanced syntactic structure enclosed in round brackets;

begin
if ba / lbracket then
  fall(103);
label10 :;
out(in);
mb;
if ba = rbracket then
begin
 copystrings;
goto label12
end;
if ba = lbracket then
begin
 copyroundbrackets;
goto label12
end;
if ba / rbracket then
 goto label12;
end copy round brackets;
procedure block(inner);
value
inner;
boolean
inner;

comment block edits a block. If the parameter 'inner' is true then the
block being edited is a statement of the enclosing block and its declarations
and statements start with an extra tab. But if the value of 'inner' is
false then the block to be edited is a part of a conditional
or for statement and extra tabs are unnecessary.

end comments are copied except that newline symbols are ignored
and tab symbols are replaced by a space.

Dumpy statements which are not followed by end are put on
a line by themselves;

begin
out(begin);
bctr := bctr + 1;
depth := depth + 1;
tabs(depth) := tabs := tabs + (if inner then 1 else 0);
newline;
nb;
specificationordeclarations(true);
label!!;

if letterordigit then
  begin
    possible(label(true));
goto label!!
  end
if be = begin then
  block(true)
else if be = comment then
  begin
    commentstatement;
goto label!!
  end
else if be = library then
  begin
    calllibrary;
goto label!!
  end
else
  statement;
tabs := tabs(depth);
if be = semantic then
  begin
    out(semantic);
nb;
  if be /= and then
    newline;
goto label!!
  end;
if be ≠ "end then
  fail(10);
if inner then
  tabs := tabs - 1;
nextline;
out(end);
exit := exit + 1;
depth := depth - 1;
if depth = 0 then
  begin
nextline;
out(endmessage);
nextline;
if 14 = 120 then
  begin
  nbe;
  if be ≠ endmessage then
    fail(10);
  end
  else
    begin
  label10 :=;
  readsymbol;
  if be = newline then
    goto label10;
  if be = tab then
    begin
    out(space);
    goto label10;
  end;
  if be ≠ "end and be ≠ else and be ≠ semicolon then
    begin
    out(be);
    goto label10;
  end
  end
end label:
procedure call library;

  comment 'call library' edits a library call by assuming that the
  call is followed by a comment;

begin
  ncb;
  if bs / comment then
    fail(vcb);
  bs := library;
  comment statement;
  out(comment);
  label(13);
  if buffer(10) / semicolon then
    begin
    le := le + 1;
    goto label(13)
    end;
    le := le + 1;
  newline;
  newline;
  end call library;

procedure codebody;

  comment 'code body' edits a procedure body written in HD99 User-code;

begin
  integer
    1,  
    j;
  scan(semicolon, semicolon);
  bs := newline;
  label(15);
  if bs = newline then
    begin
    newline;
    nba;
    i := space;
    if bs = 23 or bs = 27 or bs = 49 or bs = 53 then
      begin
      i := bs;
      nba;
      end;
      if bs $ newline then
        begin
        newline;
        if 1 $ space then
          begin
          newline;
          out(1);
          end;
          else
            begin
            out(tab);
            for j := 1 step 1 until i = spacelse:
              out(tabmax);
            if 1 $ space then
              out(1);
            end;
        end;
      goto label(15);
    end
else if bs = library then
  begin
  newline;
  out(library);
  nbs;
  if bs /= lbracket then
    fail(111);
  label16 :;
  nbs;
  if bs /= rbracket then
    begin
    out(bs);
    goto label16
    end;
  i := 1;
  out(semicolon);
  newline;
  out(lbracket);
  buffer[i] := rbracket;
  i := i + 1;
  nbs;
  if bs /= semicolon then
    fail(112);
  out(bs);
  newline;
  nbs;
  goto label15
  end
else if bs /= alg1 then
  begin
  out(bs);
  if le > linelimit then
    clearfullline(semicolon);
    reasymbo;
    goto label15;
  end;
  if le /= (tab + 1) * tabspace + 1 then
    newline
else
  i := i + 1;
  out(alg1);
  nbs;
  end; // code body;

*comment* The declarations in alg1 end end here;

*comment* Assign a suitable value to each of the variables representing an
comment: Assign a suitable value to each of the variables representing an
           Algol basic symbol. This section of 'algol edit' is machine dependent;
capitals := 12;
end := 147;
array := 72;
becomes := 161;
begin := 140;
boolean := 67;
colen := 185;
comma := 166;
comment := 128;
divide := 161;
do := 214;
else := 165;
end := 156;
equals := 152;
eqv := 155;
false := 205;
for := 134;
goto := 136;
greaterthan := 194;
gtequal := 178;
if := 133;
imp := 179;
intdiv := 145;
integer := 66;
label := 121;
lebracket := 132;
lbrace := 137;
ltbracket := 141;
lessthan := 130;
ltequal := 146;
minus := 205;
multiply := 177;
noline := 160;
nine := 9;
not := 131;
notequal := 213;
or := 163;
own := 143;
plus := 193;
procedure := 80;
real := 65;
rebracket := 118;
rebrace := 153;
ratebracket := 157;
semicolon := 152;
space := 156;
step := 182;
string := 129;
subscription := 10;
switch := 58;
tab := 174;
then := 149;
tree := 201;
tuple := 194;
tuple continue := 198;
value := 159;
while := 150;
zero := 0;
small := 63;
comment Assign suitable values to the variables representing pseudo basic symbols:

algol := 100;
andmessage := 100;
exit := 200;
keywords := 176;
library := 200;
segment := 204;
tablename := 258;

comment Assign suitable values for the elements of 'spn' and 'spn' arrays. This part of 'algol edit' is partly a matter of taste:

for i := 0 step 1 until 255 do
spn[i] := spn[i] := 0;
for i := plus,
minus,
multiply,
divide,
indent,
less-than,
less-or-equal,
equal,
greater-than,
greater-or-equal,
not-equal,
and,
or,
not,
then,
else,
colon,
eqv,
imp,
step,
until,
while do
spn[i] := spn[i] := 1;
for i := real,
integer,
boolean,
procedure,
comment,
if,
for,
goto,
end,
returndo,
comma,
semicolon,
switch do
spn[i] := 1;
for i := do,
leftbracket do
spn[i] := 1;
for i := keyword,
library,
segment do
spn[i] := 1;
comment Assign the initial values to the global variables of 'algol edit';

let r := depth := entry := 0;
bs := 1;
l := 1;
line_number := tags := 0;
tab_space := 0;
if id = 100 then
  begin
    findprog(EU, disbuffer);
    printtitle(od);
  end;
nextline;
start :=
next;
if bs /= begin then
  begin
    out(bs);
    goto start
  end;
nextline;
block(true);
end algol edit;
integer
    cases,
    1,
    id,
    no,
    out;
open(20);
out := read(20);
cases := read(20);
if (out = 10 or out = 30) and cases > 0 then
    open(out)
else
    begin
        open(30);
        writeln(30, "error in initial data");
        close(30);
        goto book2
    end;
for i := 1 step 1 until cases do
    begin
        no := read(20);
        id := read(20);
        if (id = 20 or id = 120) and no > 0 then
            begin
                algodit(no, id, out, book);
                gap(out, if out = 10 then 206 else 1);
            end
        else
            begin
                writeln(out, "error in data");
                goto book
            end;
    book2;
    close(out);
    book2;
    close(20);
end
<table>
<thead>
<tr>
<th>Distribution</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Library</td>
<td>(60)</td>
</tr>
<tr>
<td>General Distribution</td>
<td>(100)</td>
</tr>
<tr>
<td></td>
<td>(160)</td>
</tr>
</tbody>
</table>