See note inside cover

CCU 6

April 1969

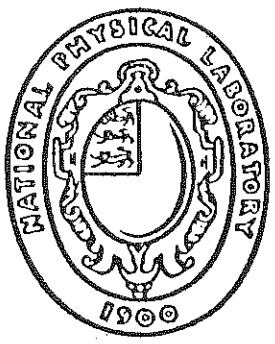
NATIONAL PHYSICAL LABORATORY

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

Approved on behalf of Director, NPL by
Dr. E.T. Goodwin, Acting Head of Central Computer Unit

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

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 characer 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
JXRS810--KPU;
PROGAREA 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; [<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; [JXRS810--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 DE021BIN.

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:-

'n'	Reason
100, 106, 109	The value of 'length of output line' is too small.
108, 111	A library call is not followed by a comment.
110	The number of 'begin's is not equal to the number of 'end's.
107, 112	There is some other error in the Algol program.
113	One of the lines in a comment is too long.
Any other number	You have found an error in SOAP, please contact the author so that it may be put right.

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

A message 'FAIL n IN FINDPROG' is output

'n'	Reason
1	Invalid character inside the string on the datatape.
2	Not twelve characters inside the string.
3,4	Error in SOAP
5	Program to be edited is not on the disc.
10,11	Prompt error.

(4) Failures while reading the Algol program from the disc

A message 'FAIL n IN READPROG' 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;
    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. Wichmann 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.

```
/...../...../...../...../...../...../... 1../...../...../...../

→ESTABLISH    JXRS810034APU;;
O/P8→

begin

        library A0006,A0013;

        comment A0006,A0013;


        library A0012;

       comment A0012;

       procedure findprog(dv, a);
               value
                       dv;
              integer
                       dv;
              integer array
                       a;
               KDF9 8 / 9 / 4 / 5;
                       V0=B 43 21 04 24 06 20 74 33;
                       V1=B 07 21 50 22 47 20 00 00;
                       V2=B 07 02 64 02 00 00 00 00;
                       V3 = Q 0 / 3 / 1;
                       SETAY0; [a]; DUPD; SHL-16; +; =M7;
                       MOM7; ABS; SET640; -; J8<Z;
                       SHL-32; +; DUP; =V5P601;
                       DUP;  V3;  +;  =V3P602;
                       DUP;  SHL+16;  +;  SET639;  +;  =V2P602;
                       [dv];
                       JS4P295; SETB215;
              1;       JS6P295; J1≠; J1≠; SET12; =C7;
                       ERASE;
                       ZERO; ZERO;
              7;       JS6P295; DUMMY; DUMMY; DUMMY; SETB235; J6=;
                       SETB321; J2=;
                       DUP; SETB45; -; J4>Z;
                       DUP; SETB14; -; J5≥Z;
                       DUP; SETB12; -; J4≥Z;
                       SETB20; +;
              3;       DC7; PERM; SHLD+6; PERM;
                        OR; REV; J7C7NZ; JS6P295;
                       DUMMY; DUMMY; DUMMY;
                       SETB235; NEV; J6≠Z;
                       JS8P295;
                       V2; OR; JSP602; J9;  EXIT;
              2;       ERASE; SETB36; J3;
              5;       SETB25; +; J3;
              8;       SET3; J9;
              4;       SET1; J9;
              6;       SET2;
              9;       SETAV0; SET12; C7; -; CAB;
                       JP299;
               ALGOL;
```

```
/...../...../...../...../...../...../... 2../...../...../...../

integer procedure readprog;
        KDF9 8 / 4 / 4 / 4;
                V0=B 43 21 64 20 03 00 74 33;
                V1=B 07 21 50 20 47 20 00 00;
                JSP601; J1; EXIT;
        1;      SETAV0; ZERO; CAB; JP299;

        P601V6;   ( finds algol basic symbols one at a time );
                V0=Q0/3/1;
                V1=Q0/1/2;
                V2=0;
                V4=-1;
                V5; =Q12;
                SET80; =M11;
                V0;     =Q15;     V1;     =Q14;
                M15; =C13;
                V4;     J1=Z;
        98;     SET 4;     EXIT 1;
        99;     SET 10;     EXIT 1;
        100;  V4;     NOT;     =V4;     EXIT 1;
        3;    ERASE;
        1;    V2;     J10≠Z;     J10C15NZ;     J6C14NZ;
        9;    COTOQ13;     V3;     DUP;     J100=Z;
                JS20P602;     J99;     JS4;     MOM12;
                JS20P602;     J2;
        13;   =V3;     MOM12N;     =RC14;
                M12; NOT; NEG; NOT; NEG;     =M14;     J6;
        2;      ERASE; M11M12; =V3; M11M12N;
                =RC14;     M12; SET82; +;     =M14;
        6;    MOM14Q;     DUP;     =V2;
                SET 6;     =C15;     J8C13NZ;
        5;    DUP;     ZERO;     SHLD+8;
                SHL+43;     SHL-43;     =C13;
                =V2;     SHL+2;     J7<Z;
                DC15;     J10;
        8;    DC13;     J5C13Z;     ERASE;
        10;   V2;     ZERO;     SHLD+8;     DC15;
                ZERO; =V6;
                REV;     =V2;     SET B377;     J3=;
                SETB276;  J14=;
        12;     C13; =M15;
                Q15;     =V0;     Q14;
                =V1;     EXIT 2;
        14;     ZERO;  NOT;  =V4;
                ZERO;  =V2;  ZERO;  =V1;  ZERO;  =V0;  EXIT 2;
        11;     SETB320; ZERO; NOT; =V6;
                ZERO; =V2; COTOQ15; J12;
        7;    DC13;     C13;     DUP;     NEG;     =+C14;
                =+M14; COTOQ13; V6; J11=Z; J6C14NZ; J9;
        4;    ZERO;     SHLD+16;     SHL+32;
                V2P602;     OR;     REV;     SHL+16;
                SET 32;     OUT;     EXIT 1;
```

```
P602V4;      (Finds identifier in Prompt Text Index and address of text);
       (VOand V1 store identifiers);
       V2=Q0/0/639;
       V3=Q212/3/1;
       V4=B 7003 7400 0000 0077;
       V5P601; =Q13;
       SET639; =M12;
       V4P601;    J96=Z;
       V2;    ZERO;
3;    SET 32;    OUT;    SET 212;
       MOM13; SHL-32; -; V3;
       =Q15;    =C15;
       J1;
96;    SET 4;  EXIT1;
99;    SET 10;  EXIT1;
200;   SET 5;  EXIT1;
201;  SET 11;    EXIT 1;
1;     E2M15; SHL+16; J4<Z; DUPD;
       MOM15; JS100; J2≠Z;
       MOM15N;    JS100;    J10=Z;    ZERO;
2;     ERASE;
4;     M+I15; DC15; J1C15NZ;
       M12M13; DUP; J200=Z; JS20;
       J201;    ZERO;    SHLD+16;    SHL+32;
       V2;    OR;    REV;    SHL+16;    J3;
10;   ZERO;    =V4P601;
       E2M15;    DUP;    =V3P601;
       MOM15;  =VO;  MOM15N;  =V1;
       PERM;    ERASE;    ERASE;    J99=Z;    EXIT 2;
100;  REV;    DUP;    SET 8;    =RC14;
101;  ZERO;    SHLD+6;    SET B36;
       MAX;    SIGN;    SHL-42;
       OR;    DC14;    J101C14NZ;
       PERM;    NEV;    AND;    VR;
21;    EXIT 1;
20;   DUP;    V4;    AND;    J21≠Z;    EXIT 2;
 ALGOL;
```

```
/...../...../...../...../...../...../... 4../...../...../.. .../

        procedure printtitle(od);
                value
                        od;
                integer
                        od;
                 KDF9 7 / 10 / 2 / 4;
                        V0 = B 0702 3275 4563 6441; ( cn crlf ul em e s t a );
                        V1 = B 4254 5163 5077 7777; ( b l i s h );
                        V2 = 0;
                        V3 = 0;
                        V4 = B 3434 0257 1760 3075; ( sc sc crlf o / p 8 em );
                        SET 5;  =RC6;  [od];  JS12P295;
                        VOP602;  SHL24;  SHL-24;  =V2;  V1P602;  =V3;
                2;      VOM6Q;  SET8;  =C7;
                1;      DC7;  ZERO;  SHLD6;  JS13P295;  J1C7NZ;  ERASE;  J2C6NZ;
                        JS16P295;  EXIT;
                 ALGOL;

        procedure print(s, n);
                value
                        n;
                string
                        s;
                integer
                        n;
                begin
                writetext(out, s);
                write(out, format( [ ssndd ] ), n);
                end  print;
```

/...../...../...../...../...../...../... 5../...../...../...../

```
procedure algoledit(linelimit, id, od, failure);
        value
                linelimit,
                id,
                od;
        integer
                linelimit,
                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 tabs' - 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
                 's' - this array is used as a temporary store when a line must be split
                  because it is too long
                 'spa', 'spb' - these arrays are used to specify the number of spaces which
                  are to be output before and after each different basic symbol;

        integer array
                ntabs[1 : 25],
                buffer,
                s[0 : 150],
                discbuffer[0 : if id = 120 then 640 else 1],
                spa,
                spb[0 : 255];
```

comment At any point in the Algol program being edited, these integer variables have the following values:-

'b ctr', 'e ctr' - the number of begins and ends that have occurred so far

'bs' - the current basic symbol

'depth' - the current nested block depth, ie. 'b ctr' - 'e ctr'

'i' - this is used as the controlled variable in a for statement

'lc' - the next symbol to be output will be put in 'buffer[lc]'

'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'. Its 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

'tabs' - 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
        bctr,
        bs,
        depth,
        ectr,
        i,
        lc,
        linenumber,
        startofstring,
        tabs,
        tabspace;
```

comment Each one of these variables represents an Algol basic symbol and has an appropriate constant value;

integer
capitala,
and,
array,
becomes,
begin,
boolean,
colon,
comma,
comment,
divide,
do,
else,
end,
equals,
eqv,
false,
for,
goto,
greaterthan,
gtequal,
if,
imp,
intdiv,
integer,
label,
lrbracket,
lsqbracket,
lstrbracket,
lessthan,
ltequal,
minus,
multiply,
newline,
nine,
not,
notequal,
or,
own,
plus,
procedure,
real,
rrbracket,
rsqbracket,
rstrbracket,
semicolon,
space,
step,
string,
subscriptten,
switch,
tab,
then,
true,
until,
value,
while,
zero,
smallz;

```
comment  Each one of these variables represents a KDF9 pseudo basic symbol and has the appropriate
         constant value;

integer
        algol,
        endmessage,
        exit,
        kdf9,
        library,
        segment,
        tabdummy;

comment   A list of the procedures in algol edit.
         read symbol
         out line ( integer value od,  integer array buffer,  integer value lc )
         nbs
         boolean letter or digit
         identifier or label
         scan ( integer value symbol 1,  integer value symbol 2 )
         next line
         comment statement
         out ! ( integer value char )
         out ( integer value char )
         clear full line ( integer value symbol )
         specifications or declarations ( boolean value declarations )
         proc declaration
         fail ( 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
         ;
```

```
/...../...../...../...../...../...../... ../...../.. . ../...../

comment    label numbers
           1  statement
           2  scan
           3  specifications or declarations
           4  clear full line
           5  copy square brackets
           6  fail
           7  nbs
           8  read string
           9  comment statement
          10  block
          11  block
          12  copy round brackets
          13  call library
          14  specifications or declarations
          15  code body
          16  code body
          17  out
          18  identifier or label
          19  expression
          20  expression
          21  for variable and list
          22  statement
          ;

procedure readsymbol;

        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(114);
        bs :=
              if id = 120 then
                    readprog
              else
                    inbasicsymbol(id);
        end  readsymbol;

procedure outline(od, buffer, lc);
        value
              od,
              lc;
        integer
              od,
              lc;
        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 'lc' of the array 'buffer'.  'outline' is a
                  machine dependent procedure;
```

/...../...../...../...../.. ..../...../.. 10../...../...../...../

```
KDF9 6 / 10 / 2 / 7;
     ( V0 = od );
     [ buffer ]; =Q7; [ lc ]; =C7; SET1; =I7; M+I7; ( Q7= lc / 1 / buffer [i] );
     [ od ]; DUP; =V0; JS12P295;
11;  J1C7Z;
10;  YOM7Q;
     SET 254; (tabdummy); J8=;
     SET 174; (tab); J9=;
     SET 176; (kdf9); J2=;
     SET 192; ( algol ); J3=;
     SET 208; ( library ); J4=;
     SET 224; ( segment ); J5=;
     SET 240; ( exit ); J6=;
     SET 190; ( em ); J7=;
     SET 181; ( becomes ); J12=;
13;  JS14P295; J10C7NZ;
1;   JS16P295; EXIT;
8;   ERASE; V0; SET 10; -; J11=Z; SET 158; ( space ); J13;
9;   V0; SET 10; -; J13=Z; ERASE; SET 158; (space); J13;
12;  V0; SET 30; -; J13=Z;
     ERASE;
     SETB117; SHC-8; JS116P295; SETB125; SHC-8; JS116P295; J11;
2;   V1; SET4; J20;
3;   V2; SET 5; J20;
4;   V3; SET 7; J20;
5;   V4; SET 7; J20;
6;   V5; SET 4; J20;
7;   V0; SET 30; J23≠;
     ERASE; V6; SET 2; J20;
20;  =RC6; V0; SET30; J21≠;
     ERASE; SETB15; JS13P295;
22;  DC6; ZERO; SHLD6; JS13P295; J22C6NZ;
     ERASE; ERASE; J10C7NZ; J1;
21;  SET 10; J24≠;
     ERASE; C6; SET6; -; J25≥Z;
     ZERO; J26;
25;  SET 1;
26;  SHC-2; JS116P295;
27;  DC6; SETB32; JS13P295; ZERO; SHLD6; JS13P295; J27C6NZ;
     ERASE; ERASE; J10C7NZ;
     J1;
24;  REV; ERASE; J28;
23;  SET10; J28≠;
     ERASE; ERASE; SETB75; JS13P295; J1;
28;  SETAV7; REV; SET 1; JP299;
     V1 = B 5344 4631 0000 0000; ( kdf9 );
     V2 = B 4154 4757 5400 0000; ( algol );
     V3 = B 5451 4262 4162 7100; ( library );
     V4 = B 6345 4755 4556 6400; ( segment );
     V5 = B 4570 5164 0000 0000; ( exit );
     V6 = B 4555 0000 0000 0000; ( em );
     V7 = B 432 15 040 077 16 400; ( [ out ] );
ALGOL;
```

```
procedure nbs;

        comment     'nbs` assigns the next non-printing symbol of the Algol program
                    being edited to the variable 'bs`;

        begin
    label7 :;
        readsymbol;
        if bs = space or bs = newline or bs = tab then
                goto label7;
        end  nbs;

boolean procedure letterordigit;
        letterordigit := (bs ≥ capitala and bs ≤ smallz) or (bs ≥ zero and bs ≤ 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(105);
    label18 :;
        out(bs);
        nbs;
        if letterordigit then
                goto label18;
        end  identifier or label;

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 bs = lrbracket then
                begin
                copyroundbrackets;
                goto label2
                end;
        if bs = lsqbracket then
                begin
                copysquarebrackets;
                goto label2
                end;
        out(bs);
        if bs ≠ symbol1 and bs ≠ symbol2 then
                begin
                nbs;
                goto label2;
                end
        end   scan;
```

```
procedure nextline;

    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
            i,
            j;
    if linenumber = 0 then
            begin
            if od = 10 then
                    begin
                    outline(od, buffer, lc - 1);
                    gap(od, 50);
                    end
            else
                    begin
                    buffer[lc] := newline;
                    outline(od, buffer, lc);
                    end;
             linenumber := 31;
            end
    else
            begin
            linenumber := linenumber - 1;
            buffer[lc] := newline;
            outline(od, buffer, lc);
            end;
    lc := tabspace × tabs;
    for i := 1 step tabspace until lc do
            begin
            buffer[i] := tab;
            for j := i + 1 step 1 until i + tabspace - 1 do
                    buffer[j] := tabdummy;
            end  for i;
    lc := lc + 1;
    end  nextline;
```

```
procedure commentstatement;

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

        begin
        nextline;
        tabs := tabs + 1;
        out(bs);
    label9 :;
        if lc > 150 then
                fail(113);
        readsymbol;
        if bs ≠ semicolon then
                begin
                if bs = newline then
                        nextline
                else if bs = tab then
                        out1(space)
                else
                        out1(bs);
                goto label9;
                end;
        out1(semicolon);
        tabs := tabs - 1;
        nextline;
        nextline;
        nbs;
        end  comment statement;

procedure out1(char);
        value
                char;
        integer
                char;

        comment   'out 1' inserts the symbol 'char' into the output buffer and increases the
                  counter 'lc' by one;

        begin
        buffer[lc] := char;
        lc := lc + 1
        end  out 1;
```

```
procedure out(char);
        value
                char;
        integer
                char;

        comment   '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;

        begin
        if buffer[lc - 1] = subscriptten then
                begin
                if char = plus or char = minus then
                        begin
                        out1(char);
                        nbs;
                        out1(bs);
                        goto label17
                        end
                end;
        if spb[char] ≠ 0 and buffer[lc - 1] ≠ space then
                out1(space);
        if (char = comma or char = semicolon) and buffer[lc - 1] = space then
                lc := lc - 1;
        out1(char);
        if spa[char] ≠ 0 then
                out1(space);
    label17 :;
        if lc > linelimit + 3 then
                fail(106);
        if lc ≥ linelimit then
                clearfullline(space);
        end  out character;
```

```
procedure clearfullline(symbol);
        value
                symbol;
        integer
                symbol;

        comment    clear full line' is called when an edited line is too long to be
                   output on a single line.  It looks for the latest occurrence of the
                   symbol specified by the parameter 'symbol', and splits the line at
                   this point.  It outputs the first part of the line and puts the
                   remaining characters at the beginning of the next line.  The
                   procedure fails if it is unable to find a point at which the line
                   can be split;

        begin
        integer
                j,
                k;
        j := lc - 1;
        for k := 1 step 1 until linelimit - tabspace × (tabs - 1) + 1 do
                begin
                if buffer[j] = symbol then
                        goto label4;
                s[k] := buffer[j];
                j := j - 1;
                end;
        fail(100);
    label4 :;
        lc := j;
        if symbol = semicolon then
                out1(semicolon);
        tabs := tabs + 1;
        nextline;
        lc := lc - 1;
        for j := 1 step 1 until k - 1 do
                buffer[lc + j] := s[k - j];
        lc := lc + k;
        tabs := tabs - 1;
        end  clear full line;
```

```
procedure specificationsordeclarations(declarations);
        value
                declarations;
        boolean
                declarations;

        comment   If the parameter of 'sepcifications or declarations' 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
    label3 :;
        if bs = procedure and declarations then
                begin
                procdeclaration;
                goto label3;
                end
        else if bs = switch and declarations then
                begin
                scan(becomes, becomes);
                nbs
                end
        else if bs = library then
                begin
                calllibrary;
                goto label3
                end
        else if bs = comment then
                begin
                commentstatement;
                goto label3;
                end
        else if bs = real or bs = integer or bs = boolean or bs = array or bs = switch or bs = label or bs =
                string or bs = own or bs = value or bs = procedure then
                begin
                out(bs);
                nbs;
                goto label3;
                end;
        if lc ≠ tabspace × tabs + 1 then
                begin
                tabs := tabs + 1;
                nextline;
           label14 :;
                scan(comma, semicolon);
                if bs = semicolon then
                        tabs := tabs - 1;
                nextline;
                if bs = comma then
                        begin
                        nbs;
                        goto label14;
                        end;
                nbs;
                goto label3;
                end;
        end  specifications or declarations;
```

```
procedure procdeclaration;

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

    begin
    scan(semicolon, semicolon);
    tabs := tabs + 1;
    nextline;
    nbs;
    specificationsordeclarations(false);
    if bs = segment then
            scan(semicolon, semicolon)
    else
            begin
            if bs = kdf9 then
                    codebody
            else
                    statement;
            if bs = semicolon then
                    out(bs)
            else
                    fail(107);
            end;
     tabs := ntabs[depth];
     nextline;
     nextline;
     nbs;
     end  proc declaration;
```

```
procedure fail(n);
        value
                n;
        integer
                n;

        comment  '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  code body
                 113  comment statement
                 114  read symbol
                 114  for variable and list
                 117  if clause
                 ;

        begin
        lc :=
                if lc > linelimit then
                        linelimit
                else
                        lc;
        nextline;
        print( [ fail ], n);
    label6 :;
        readsymbol;
        if bs = begin then
                bctr := bctr + 1
        else if bs = end then
                ectr := ectr + 1;
        if bctr = ectr then
                goto failure
        else
                goto label6;
        end  fail;
```

```
procedure expression(symbol);
        value
                symbol;
        integer
                symbol;

        comment  This procedure edits a conditional or simple expression.  It
                 stops when the current basic symbol is end or 'comma' or  step
                 or  while  or 'symbol';

        begin
        if bs = if then
                begin
                tabs := tabs + 1;
                nextline;
           label19 :;
                ifclause;
                expression(else);
                tabs := tabs - 1;
                nextline;
                out1(else);
                nbs;
                if bs = if then
                        begin
                        out1(space);
                        goto label19
                        end;
                tabs := tabs + 1;
                nextline;
                expression(symbol);
                tabs := tabs - 2
                end
         else
                begin
            label20 :;
                if bs = lrbracket then
                        copyroundbrackets
                else if bs = lsqbracket then
                        copysquarebrackets;
                if bs = end or bs = comma or bs = step or bs = while or bs = symbol then

                else
                        begin
                        out(bs);
                        nbs;
                        goto label20
                        end;
                end;
         end  expression;
```

```
procedure forvariableandlist;

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

        begin
        if bs ≠ for then
                fail(114);
        tabs := tabs + 1;
        scan(becomes, becomes);
    label21 :;
        nbs;
        scan(comma, do);
        nextline;
        if bs = comma then
                goto label21;
        nbs
        end  for variable and list;

procedure ifclause;

        comment   This procedure edits an  if clause ;

        begin
        if bs ≠ if then
                fail(117);
        out(bs);
        nbs;
        expression(then);
        out(bs);
        tabs := tabs + 1;
        nextline;
        nbs;
        end  if clause;
```

```
procedure statement;

    comment    `statement` edits any unlabelled statement;

    begin
label1 :;
    if letterordigit then
            begin
            possiblelabel(false);
            goto label1
            end;
    if bs = if then
            begin
            ifclause;
            goto label1
            end;
    if bs = for then
            begin
            forvariableandlist;
            goto label1
            end;
    if bs = begin then
            block(false);
label22 :;
    if bs = lrbracket then
            copyroundbrackets
    else if bs = lsqbracket then
            copysquarebrackets
    else if bs = if then
            expression(semicolon)
    else if bs = else then
            begin
            tabs := tabs - 1;
            nextline;
            out1(else);
            nbs;
            if bs = if then
                    out1(space)
            else
                    begin
                    tabs := tabs + 1;
                    nextline;
                    end;
            goto label1
            end;
    if bs ≠ semicolon and bs ≠ end then
            begin
            out(bs);
            nbs;
            goto label22
            end
    end  statement;
```

```
procedure possiblelabel(insertingadummystatementispossible);
    value
        insertingadummystatementispossible;
    boolean
        insertingadummystatementispossible;

    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
        i,
        di;
    identifierorlabel;
    if bs = colon then
        begin
        if tabs ≠ 0 then
            begin
            di := (tabspace + 1) + 2;
            lc := lc - di;
            for i := tabspace × tabs + 1 - di step 1 until lc - 1 do
                buffer[i] := buffer[i + di];
            for i := tabspace × (tabs - 1) + 1 step 1 until tabspace × tabs - di do
                buffer[i] := space;
            end;
        out(colon);
        nbs;
        if bs ≠ semicolon then
            begin
            if insertingadummystatementispossible then
                begin
                out(semicolon);
                nextline
                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 newline symbol;

        begin
        if buffer[lc - 1] ≠ space then
                outl(space);
        startofstring := lc;
        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 newline symbol;

        begin
        integer
                i;
        if bs ≠ lstrbracket then
                fail(101);
    label8 :;
        outl(bs);
        if lc > linelimit then
                begin
                lc := startofstring;
                tabs := tabs + 1;
                nextline;
                tabs := tabs - 1;
                if lc ≥ startofstring then
                        fail(109);
                startofstring := lc;
                for i := startofstring step 1 until linelimit do
                        outl(buffer[i]);
                end;
        readsymbol;
        if bs = lstrbracket then
                begin
                readstring;
                goto label8;
                end
        else if bs = newline then
                begin
                tabs := tabs + 1;
                nextline;
                tabs := tabs - 1;
                startofstring := lc;
                nbs;
                goto label8;
                end
        else if bs ≠ rstrbracket then
                goto label8;
        end  read string;
```

```
procedure copysquarebrackets;

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

      begin
      if bs ≠ lsqbracket then
            fail(102);
label5 :;
      out(bs);
      nbs;
      if bs = lrbracket then
            begin
            copyroundbrackets;
            goto label5
            end;
      if bs = lsqbracket then
            begin
            copysquarebrackets;
            goto label5
            end;
      if bs ≠ rsqbracket then
            goto label5;
      end  copy square brackets;

procedure copyroundbrackets;

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

      begin
      if bs ≠ lrbracket then
            fail(103);
label12 :;
      out(bs);
      nbs;
      if bs = lstrbracket then
            begin
            copystring;
            goto label12
            end;
      if bs = lrbracket then
            begin
            copyroundbrackets;
            goto label12
            end;
      if bs ≠ rrbracket 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.

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

        begin
        out(begin);
        bctr := bctr + 1;
        depth := depth + 1;
        ntabs[depth] := tabs := tabs + (if inner then 1 else 0);
        nextline;
        nbs;
        specificationsordeclarations(true);
    label11 :;
        if letterordigit then
                begin
                possiblelabel(true);
                goto label11
                end;
        if bs = begin then
                block(true)
        else if bs = comment then
                begin
                commentstatement;
                goto label11
                end
        else if bs = library then
                begin
                calllibrary;
                goto label11
                end
        else
                statement;
        tabs := ntabs[depth];
        if bs = semicolon then
                begin
                out(semicolon);
                nbs;
                if bs ≠ end then
                        nextline;
                goto label11;
                end;
```

```
if bs ≠ end then
        fail(104);
if inner then
        tabs := tabs - 1;
nextline;
out(end);
ectr := ectr + 1;
depth := depth - 1;
if depth = 0 then
        begin
        nextline;
        out(endmessage);
        nextline;
        if id = 120 then
                begin
                nbs;
                if bs ≠ endmessage then
                        fail(110);
                end
        end
else
        begin
   label10 :;
        readsymbol;
        if bs = newline then
                goto label10;
        if bs = tab then
                begin
                out(space);
                goto label10
                end;
        if bs ≠ end and bs ≠ else and bs ≠ semicolon then
                begin
                out(bs);
                goto label10
                end;
        end
end  block;
```

```
procedure calllibrary;

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

        begin
        nbs;
        if bs ≠ comment then
                fail(108);
        bs := library;
        commentstatement;
        out(comment);
    label13 :;
        if buffer[lc] ≠ semicolon then
                begin
                lc := lc + 1;
                goto label13
                end;
        lc := lc + 1;
        nextline;
        nextline;
        end  call library;

procedure codebody;

        comment    'code body' edits a procedure body written in KDF9 User-code;

        begin
        integer
                i,
                j;
        scan(semicolon, semicolon);
        bs := newline;
    label15 :;
        if bs = newline then
                begin
                nextline;
                nbs;
                i := space;
                if bs = 23 or bs = 27 or bs = 49 or bs = 53 then
                        begin
                        i := bs;
                        nbs;
                        end;
                if bs ≤ nine then
                        begin
                        if i ≠ space then
                                begin
                                nextline;
                                out1(i);
                                end;
                        end
                else
                        begin
                        out1(tab);
                        for j := 1 step 1 until tabspace - 1 do
                                out1(tabdummy);
                        if i ≠ space then
                                out1(i)
                        end;
                goto label15;
                end
```

```
else if bs = library then
        begin
        nextline;
        out(library);
        nbs;
        if bs ≠ lrbracket then
                fail(111);
   label16 :;
        nbs;
        if bs ≠ rrbracket then
                begin
                out(bs);
                goto label16
                end;
        i := lc;
        out(semicolon);
        nextline;
        out(lrbracket);
        buffer[i] := rrbracket;
        lc := i + 1;
        nbs;
        if bs ≠ semicolon then
                fail(112);
        out(bs);
        nextline;
        nbs;
        goto label15
        end
else if bs ≠ algol then
        begin
        out1(bs);
        if lc ≥ linelimit then
                clearfullline(semicolon);
        readsymbol;
        goto label15;
        end;
if lc ≠ (tabs + 1) × tabspace + 1 then
        nextline
else
        lc := tabs × tabspace + 1;
out(algol);
nbs;
end  code body;

comment  The declarations in  algol edit  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;

```
capitala := 12;
and := 147;
array := 72;
becomes := 181;
begin := 140;
boolean := 67;
colon := 185;
comma := 166;
comment := 128;
divide := 161;
do := 214;
else := 165;
end := 156;
equals := 162;
eqv := 195;
false := 205;
for := 134;
goto := 136;
greaterthan := 194;
gtequal := 178;
if := 133;
imp := 179;
intdiv := 145;
integer := 66;
label := 121;
lrbracket := 132;
lsqbracket := 137;
lstrbracket := 141;
lessthan := 130;
ltequal := 146;
minus := 209;
multiply := 177;
newline := 160;
nine := 9;
not := 131;
notequal := 210;
or := 163;
own := 143;
plus := 193;
procedure := 80;
real := 65;
rrbracket := 148;
rsqbracket := 153;
rstrbracket := 157;
semicolon := 152;
space := 158;
step := 182;
string := 122;
subscriptten := 10;
switch := 88;
tab := 174;
then := 149;
true := 221;
until := 198;
value := 159;
while := 150;
zero := 0;
smallz := 63;
```

```
comment  Assign suitable values to the variables representing pseudo basic symbols;

algol := 192;
endmessage := 190;
exit := 240;
kdf9 := 176;
library := 208;
segment := 224;
tabdummy := 254;

comment  Assign suitable values for the elments of 'spa' and 'spb' arrays.
         This part of 'algol edit' is partly a matter of taste;

for i := 0 step 1 until 255 do
        spb[i] := spa[i] := 0;
for i := plus,
        minus,
        multiply,
        divide,
        intdiv,
        lessthan,
        ltequal,
        equals,
        gtequal,
        greaterthan,
        notequal,
        becomes,
        and,
        or,
        not,
        then,
        else,
        colon,
        eqv,
        imp,
        step,
        until,
        while do
        spb[i] := spa[i] := 1;
 for i := real,
        integer,
        boolean,
        procedure,
        comment,
        if,
        for,
        goto,
        own,
        end,
        rstrbracket,
        comma,
        semicolon,
        switch do
        spa[i] := 1;
 for i := do,
        lstrbracket do
        spb[i] := 1;
 for i := kdf9,
        library,
        segment do
        spa[i] := 1;
```

```
        comment   Assign the initial values to the global variables of 'algol edit';

        bctr := depth := ectr := 0;
        bs := - 1;
        lc := 1;
        linenumber := tabs := 0;
        tabspace := 6;
        if id = 120 then
                begin
                findprog(20, discbuffer);
                printtitle(od);
                end;
        nextline;
start :;
        nbs;
        if bs ≠ begin then
                begin
                out(bs);
                goto start
                end;
        nextline;
        block(true);
        end  algoledit;
```

```
        integer
                cases,
                i,
                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);
                writetext(30, [ [ c ] error * in * initial * data [ c ] ] );
                close(30);
                goto boob2
                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
                        algoledit(no, id, out, boob);
                        gap(out, if out = 10 then 200 else 1);
                        end
                else
                        begin
                        writetext(out, [ [ c ] error * in * data [ c ] ] );
                        goto boob
                        end;
                end;
    boob :;
        close(out);
    boob2 :;
        close(20);
end
```

Distribution

Library	(60)
General Distribution	(100)
	(160)