Program name. LIB PLOT
Source. M. N. Mitchison, DMIP; *Date of issue.* December 1968.

¶*Description.* This program plots, or tabulates, a function, or tabulates a list of functions, over a specified range, to the current output device. The functions plotted or tabulated must be functions of one argument and one result, which, in the case of PLOT, must be a number.

The user may specify, by setting the values of global variables, the format of the output.

¶*How to use the program.* Compile the program by typing:
COMPILE(LIBRARY([LIB PLOT]));
The following global variables are used by the program to format the output, and may be changed by the user:

LINELENGTH	Initial value 60	Sets the maximum length of lines output, and thus affects the scaling in PLOT.
AXPRINT	Initial value TRUE	A boolean which, when true, causes output of the axes (if possible) when plotting.
AXESMARK	Initial value "."	The item which is printed to represent points on the axis.
CURVEMARK	Initial value "X"	The item which is printed to represent points on the curve.
DOREVERSE	Initial value FALSE	When true, plotting or tabulating will be done in the reverse order from that specified.
SEPARATE	Initial value 10	Controls the number of character positions between the start of each successive column in TAB.

The function PLOT. PLOT takes four arguments, these are:
(1) The function to be plotted.
(2) The lower limit of the plot.
(3) The increment, that is, the interval between successive points on the graph.
(4) The upper limit of the plot.
For example, PLOT(SIN, –5, 0. 5, 5); will plot the function SIN from –5 to 5 in steps of 0. 5.

The function TAB. TAB takes four arguments. These are similar to PLOT except that the function argument may be replaced by a list of functions.
Tabulation is in the form: X F1 F2 F3, and so on, where X is the argument and F1, F2, etc., are the results of the functions when applied to X.

Where possible the columns are headed automatically by the name of the function being tabulated—see examples. It should be noted that the length of the list of functions to be tabulated should be less than LINELENGTH/SEPARATE-1.

¶*Method used. Plot.* First the parameters are checked to make sure they are of the correct type and that the upper bound of the range is greater than the lower one. The modulus of the interval is now calculated and if DOREVERSE is true the upper and lower bounds are reversed. The function PLOTDEAL is called. This constructs a list of the values of the function, and finds a suitable scale, taking LINELENGTH into account. This list is now used to plot the graph, the function PLOTAXPR being used to print the horizontal axis if possible. The values of the graph at the ends of the axes are printed if it is found that, after rounding them to the nearest integer, they are not zero.

Tab. After checking its parameters in a similar way to PLOT, and defining a function SOS (not global) to assign to CUCHAROUT, a loop is entered which prints out the name in the FNPROPS of the functions it is tabulating if this is not NIL. Between the limits given, it now iteratively prints out the value of the argument of the functions and then their results.

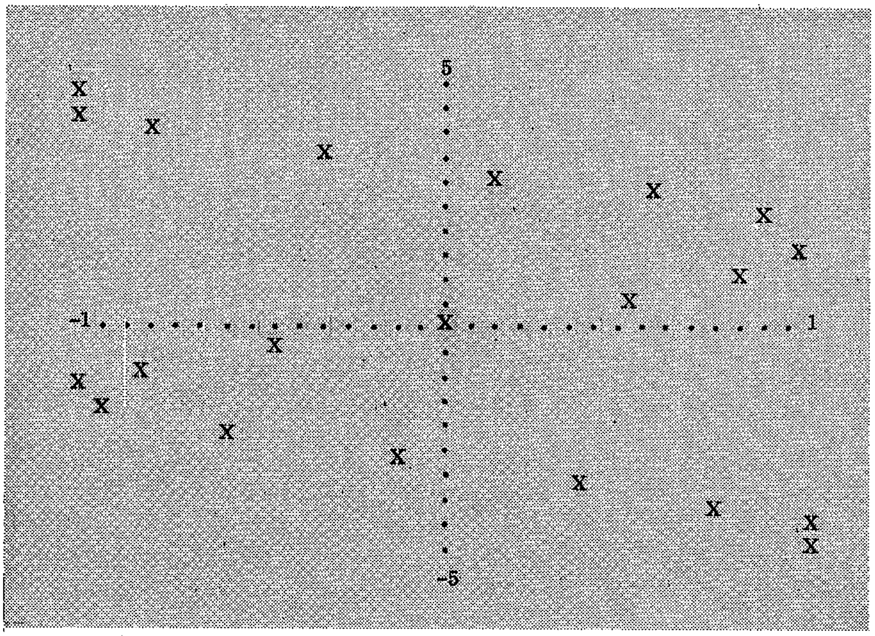
Both PLOT and TAB terminate their output with three line feeds.

¶*Global variables*. All variables used by the program are prefixed by the letters PLOT.

¶*Store used*. The program occupies some 3 blocks of core.

¶*Examples of use*

```
COMPILE(LIBRARY([LIB PLOT]));
PLOT(SIN,—5, 0.5, 5);
```



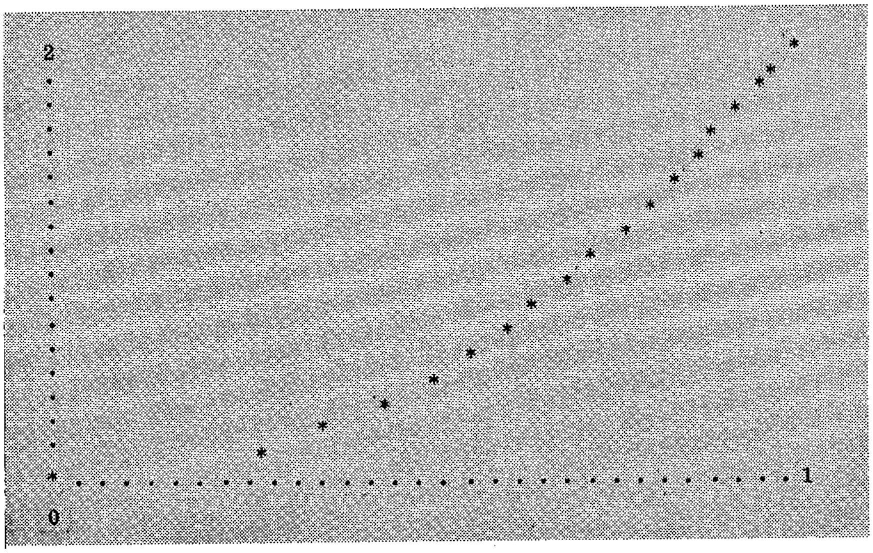
```
TAB(SIN,—5, 0.5, 5);
               SIN
—5.0         0.9589
—4.5         0.9775
—4.0         0.7568
—3.5         0.3508
—3.0        —0.1411
—2.5        —0.5984
—2.0        —0.9093
—1.5        —0.9975
—1.0        —0.8415
—0.5        —0.4794
 0           0
 0.5         0.4794
```

```
1.0       0.8415
1.5       0.9975
2.0       0.9093
2.5       0.5984
3.0       0.1411
3.5      —0.3508
4.0      —0.7568
4.5      —0.9775
5        —0.9589
```

```
TAB([% SIN, COS, TAN, LAMBDA X; X↑2—1; END, SQRT %], 3, 0.5, 9);
```

```
            SIN        COS        TAN                  SQRT
3.0       0.1411    —0.99      —0.1425     8.0       1.732
3.5      —0.3508    —0.9365     0.3746    11.25      1.871
4.0      —0.7568    —0.6536     1.158     15.0       2.0
4.5      —0.9775    —0.2108     4.639     19.25      2.121
5.0      —0.9589     0.2837    —3.38      24.0       2.236
5.5      —0.7055     0.7087    —0.9955    29.25      2.345
6.0      —0.2794     0.9602    —0.2909    35.0       2.449
6.5       0.2152     0.9766     0.2204    41.25      2.549
7.0       0.657      0.7539     0.8716    48.0       2.646
7.5       0.938      0.3466     2.707     55.25      2.739
8.0       0.9894    —0.1456    —6.795     63.0       2.828
8.5       0.7984    —0.6021    —1.326     71.25      2.915
9.0       0.412     —0.9112    —0.4522    80.0       3.0
```

```
"*" —> CURVEMARK;

PLOT(SQRT, 0, 0.1, 2);
```



```
[PLOT]


FUNCTION PLOTMOD PLOTYY;
  IF PLOTYY<0 THEN -PLOTYY ELSE PLOTYY CLOSE;
END


FUNCTION PLOTSP PLOTLL PLOTHH;
  VARS PLOTLLL PLOTSS;
  []->PLOTLLL;
 L:PLOTLL.DEST->PLOTLL->PLOTSS;
  PLOTLLL<>[%PLOTSS%]->PLOTLLL;
  PLOTHH-1->PLOTHH;
  IF PLOTHH THEN GOTO L CLOSE;
  PLOTLL;PLOTLLL;
END


VARS LINELENGTH SEPARATE AXESMARK CURVEMARK
 AXPRINT DOREVERSE PLOTAXPR;
60->LINELENGTH; "X"->CURVEMARK; "."->AXESMARK;
1->AXPRINT; 10->SEPARATE; 0->DOREVERSE;



FUNCTION PLOTDEAL PLOTHH PLOTFF PLOTLL PLOTSSS PLOTUU;
  VARS PLOTBB PLOTYY PLOTCC PLOTZZ;
  PLOTLL.PLOTFF->PLOTBB;
  PLOTBB->PLOTYY;
  [%PLOTBB%]->PLOTCC;
  IF PLOTLL.PLOTMOD<0.4*PLOTSSS THEN [%PLOTCC%]->PLOTCC CLOSE;
 L:PLOTLL+PLOTSSS->PLOTLL;
  IF PLOTHH(PLOTLL,PLOTUU)
  THEN PLOTYY; PLOTBB;
    LINELENGTH/(PLOTBB-PLOTYY)::(-LINELENGTH*PLOTYY)/(PLOTBB-PLOTYY)+0.5;
    PLOTCC;
  EXIT;
  PLOTLL.PLOTFF->PLOTZZ;
  IF PLOTZZ>PLOTBB THEN PLOTZZ->PLOTBB
    ELSEIF PLOTZZ<PLOTYY THEN PLOTZZ->PLOTYY
  CLOSE;
  IF PLOTLL.PLOTMOD<0.4*PLOTSSS THEN [%PLOTZZ%]->PLOTZZ CLOSE;
  PLOTZZ::PLOTCC->PLOTCC;
  GOTO L;
END



FUNCTION TAB PLOTFF PLOTLL PLOTSS PLOTUU;
  VARS PLOTYY PLOTBB PLOTCC PLOTZZ PLOTCOUN PLOTHH;
  IF PLOTFF.ATOM THEN TAB([%PLOTFF%],PLOTLL,PLOTSS,PLOTUU) EXIT;
  CUCHAROUT->PLOTHH;
  IF PLOTLL.ISNUMBER.NOT OR PLOTSS.ISNUMBER.NOT OR PLOTUU.ISNUMBER.NOT
    OR PLOTLL>PLOTUU THEN GOTO KK
  CLOSE;
  PLOTSS.PLOTMOD->PLOTSS;
  IF DOREVERSE THEN
     PLOTLL,PLOTUU->PLOTLL->PLOTUU; NONOP< ->PLOTZZ; -PLOTSS->PLOTSS
  ELSE
    NONOP> ->PLOTZZ
  CLOSE;

  FUNCTION PLOTSOS PLOTCC;
    VARS CUCHAROUT;
    PLOTHH->CUCHAROUT;
    IF PLOTCC=63 THEN SP(SEPARATE-PLOTCOUN);0->PLOTCOUN EXIT;
    IF PLOTCC=17 THEN 0->PLOTCOUN ELSE PLOTCOUN+1->PLOTCOUN CLOSE;
    CUCHAROUT(PLOTCC);
  END;
  0->PLOTCOUN;
  PLOTSOS->CUCHAROUT;
  PLOTFF->PLOTBB;
 L:INTOF(LINELENGTH/SEPARATE-1)->PLOTLLL;
  IF PLOTFF.LENGTH>PLOTLLL THEN PLOTFF,PLOTLLL.PLOTSP->PLOTFF->PLOTLLL CLOSE;
  IF PLOTBB.NULL THEN GOTO LO CLOSE;
  PLOTBB.HD->PLOTYY;
  IF NOT(PLOTYY.ISFUNC) THEN GOTO KK CLOSE;
  PLOTBB.TL->PLOTBB;
  FNPROPS(PLOTYY)->PLOTCC;
 LB:IF PLOTCC.ATOM.NOT THEN PLOTCC.HD->PLOTCC; GOTO LB CLOSE;
  63.CUCHAROUT;
  IF PLOTCC.ISWORD AND NOT(PLOTCC=NIL) THEN PR(PLOTCC) CLOSE;
  GOTO L;
 LO:1.NL;
  PLOTFF->PLOTBB;
  PR(PLOTLL);
  APPLIST(PLOTBB,LAMBDA PLOTYY; 63.CUCHAROUT; PLOTLL.PLOTYY.PR; END );
  PLOTLL+PLOTSS->PLOTLL;
  IF PLOTZZ(PLOTLL,PLOTUU)
   THEN IF PLOTLLL.ATOM THEN 3.NL;PLOTHH->CUCHAROUT; EXIT;
        2.NL; GOTO L;
  CLOSE;
  GOTO LO;
 KK:PR('NOT SUITABLE ARGUMENTS FOR TAB: GIVE FUNCTION, OR LIST OF MORE
    THAN ONE FUNCTION, THEN LOWER BOUND, THEN INTERVAL, THEN UPPER BOUND');
  PLOTHH->CUCHAROUT;
  .SETPOP;

END



FUNCTION PLOT PLOTFF PLOTLL PLOTSSS PLOTUU;
  VARS PLOTBB PLOTYY PLOTCC PLOTZZ PLOTCOUN PLOTHH PLOTLLL PLOTHEVAL;
  IF PLOTFF.ISFUNC.NOT OR PLOTLL.ISNUMBER.NOT OR PLOTSSS.ISNUMBER.NOT
    OR PLOTUU.ISNUMBER.NOT OR PLOTLL>PLOTUU THEN
      'NOT SUITABLE ARGUMENTS FOR PLOT: GIVE FUNCTION, THEN LOWER BOUND,
       THEN INTERVAL, THEN UPPER BOUND'.PR; .SETPOP;
  EXIT;
  PLOTSSS.PLOTMOD->PLOTSSS;
  IF DOREVERSE
   THEN PLOTUU,PLOTLL->PLOTUU->PLOTLL; -PLOTSSS->PLOTSSS;
        NONOP<
   ELSE NONOP>
  CLOSE;
  PLOTDEAL(PLOTFF,PLOTLL,PLOTSSS,PLOTUU)->PLOTBB->PLOTYY->PLOTHH->PLOTLLL;
  PLOTYY.BACK.INTOF->PLOTZZ;
  IF PLOTZZ=<LINELENGTH AND PLOTZZ>=0 AND AXPRINT
    THEN
      SP(PLOTZZ-1); PLOTUU.PR;
      APPLIST(PLOTBB,
        LAMBDA PLOTSS;
          1.NL;
          IF PLOTSS.ATOM.NOT THEN .PLOTAXPR;
             ELSE INTOF(PLOTYY.FRONT*PLOTSS+PLOTYY.BACK)->PLOTCC;
              IF PLOTCC<PLOTZZ
                 THEN PLOTCC.SP; CURVEMARK.PR;
                      SP(PLOTZZ-1-PLOTCC); AXESMARK.PR;
                 ELSEIF PLOTCC>PLOTZZ
                 THEN PLOTZZ.SP; AXESMARK.PR;
                      SP(PLOTCC-1-PLOTZZ); CURVEMARK.PR;
                 ELSE PLOTCC.SP; CURVEMARK.PR;
              CLOSE;
          CLOSE;
        END );
      1.NL; PLOTZZ.SP; PLOTLL.PR;
    ELSE APPLIST(PLOTBB,
      LAMBDA PLOTSS;
        1.NL;
        IF PLOTSS.ATOM.NOT
        THEN IF AXPRINT THEN .PLOTAXPR; EXIT
           PLOTSS.HD->PLOTSS;
        CLOSE;
        SP(PLOTYY.FRONT*PLOTSS+PLOTYY.BACK); CURVEMARK.PR;
      END )
  CLOSE;
  3.NL;
END
```

```
FUNCTION PLOTAXPR;
  INTOF(PLOTYY.FRONT*PLOTSS.HD+PLOTYY.BACK)->PLOTCC;
    FUNCTION PLOTFLOP PLOTBB;
      IF PLOTBB THEN PLOTCOUN+1->PLOTCOUN;PLOTBB.PLOTFLOP; CLOSE;
    END
  0->PLOTCOUN;
  CUCHAROUT,PLOTFLOP->CUCHAROUT->PLOTFLOP;
  INTOF(PLOTLLL+0.5)->PLOTLLL;
  IF PLOTLLL THEN PLOTLLL.PR CLOSE;
  CUCHAROUT,PLOTFLOP->CUCHAROUT->PLOTFLOP;
  INTOF(PLOTHH+0.5)->PLOTHH;
  IF PLOTHH>0
    THEN INTOF(LOG(PLOTHH)+1)
    ELSEIF PLOTHH<0 THEN INTOF(LOG(-PLOTHH)+2)
    ELSE 0
  CLOSE;
  ->PLOTHEVAL;
 L: IF PLOTCC=PLOTCOUN
      THEN CURVEMARK.PR;
      ELSEIF LOGAND(PLOTZZ-PLOTCOUN,1) THEN 1.SP; ELSE AXESMARK.PR
    CLOSE;
  PLOTCOUN+1->PLOTCOUN;
  IF PLOTCOUNT=<(LINELENGTH-PLOTHEVAL) THEN GOTO L CLOSE;
  CUCHAROUT,PLOTFLOP->CUCHAROUT->PLOTFLOP;
  IF PLOTHH THEN PLOTHH.PR CLOSE;
  PLOTFLOP->CUCHAROUT;
END;

2.NL; 'PLOT READY FOR USE`.PR; 2.NL;
```

Program name. LIB POPEDIT
Source. R. J. Popplestone, A. P. Ambler, R. Dunn, DMIP;
Date of issue. December 1968.

¶Description. This is a package which provides sequential editing (and optional compilation) of POP-2 files. The file to be edited, and the editing instructions, may come from any input device, and the edited file may be output to any number of devices.

¶How to use program. The program should be compiled by typing:
COMPILE(LIBRARY([LIB POPEDIT]));
The functions which are provided are POPGOBBLE and POPEDIT.

¶POPGOBBLE. POPGOBBLE takes a character repeater as its argument, and applies this until the end of the associated file is reached. That is,
POPGOBBLE ϵ Character repeater => ().

¶POPEDIT. POPEDIT takes three arguments. These are:
(1) The character repeater of the file to be edited.
(2) The character repeater of the edit commands.
(3) A list of character consumers of the devices to which the edited file is to be output.

The result of POPEDIT is the character repeater of the edited file. That is,
POPEDIT ϵ Character repeater, character repeater, list
=> character repeater.
When this resultant character repeater is used (for example, by COMPILE or POPGOBBLE) it produces the characters of the original file modified by the edit commands, copying the characters to the output devices as a side effect.

Examples of the use of POPEDIT and POPGOBBLE are given later in this document.

If the edit commands are being typed in from the console (that is, the character repeater for the edit commands is CHARIN) the program outputs the following message after POPEDIT has been called:
'READY: TYPE EDIT COMMANDS'
The first edit command should now be typed in, followed by carriage return/line feed. When this command has been obeyed, ':' is output and the next command should be given, and so on.

If the edit commands are being given off-line (that is, from a paper-tape or disc file), they are read automatically when required.

When the file being edited has successfully been read to its end, all the input and output files are closed, and the following message is output to the console:
'EDIT FINISHED. OUTPUT FILES CLOSED'

¶Edit commands The type of edit commands available are the same as those in the Elliott program 'EDIT41'.

An edit command consists of three parts: a function part, a space, and a string of characters. Two characters specify the function, and after the space, the remaining characters up to, but not including the next new line character form the string.

The edit 'functions' provided are:
FL DL FC DC FE DE IS IL IB SH
These have the following effects:

FL — Find line: The input file is copied until a line beginning with the edit string is found. The last character copied is the last character of the edit string.

DL — Delete to line: The input file is skipped until a new line beginning with the edit string is found. The last character skipped is the last character of the edit string.

FC — Find characters successively: If the characters of the edit string are $C_1, C_2 \ldots C_n$, then the input file is copied until C_1 has been copied, and then further until C_2 has been copied and so on until C_n has been copied.

DC — Delete to characters successively: If the characters of the edit string are $C_1 \ldots C_n$, then the input file is skipped until C_1 has been skipped, and then further until C_2 has been skipped, and so on until C_n has been skipped.

FE — Find end of line: The input file is copied up to, but not including the next newline character. The newline character is read and stored and will always be regarded as the next character from the input device.

DE — Delete to end of line: The input file is skipped up to but not including the next newline character. The treatment of this newline character is the same as for FE.

IS — Insert on same line: The edit string is copied.

IL — Insert on a new line: A newline character is output and then the edit string is copied.

IB — Insert a block: The edit string, including newline characters, is copied. The string must be terminated by a combination of a newline followed by a '↑' character. The newline and '↑' characters are not output.

SH — Reads the remainder of the input file up to its end.

¶*Ignorable characters*. Spaces are copied, but are ignored for search purposes. For example, when searching for a line which is indented, it is not necessary to put the preliminary spaces into the edit string. However, if the edit string begins or ends with a space then this space *is* significant and is not ignored for search purposes.

¶*Additional commands—on and off*. The commands ON and OFF can be inserted in the edit commands. They do not have any edit string and are used to monitor the editing process.

ON — causes the edited stream to be copied onto the current user output device in *addition* to any other devices which are being used to copy the output.

OFF — causes the copying onto the current user output device to stop.

¶*Errors*. If an unacceptable edit command is read, then the following message is output:
POPEDERR CULPRIT X
where X is the offending character.
If the edit stream is being input from the teletype, then
'TRYAGAIN'
is output and the user should retype the edit command. If, however, the edit stream was coming from another device, the message:
'CONTINUE BY TYPING EDIT COMMANDS'
is output, the edit file closed, and the edit file reverts to the teletype. The user must continue his edit by typing in the commands from the teletype.

If the end of the source file is read while any command other than SH is being obeyed, then the message:
'END OF SOURCE FILE. OUTPUT FILES CLOSED'
is output, and the edit is terminated. The user must take any necessary action to recover the original files. This error is most likely to be caused by a FL command, the string given not being correct.

If the end of the edit commands file is reached before the end of the source file is reached, the message:
'END OF EDIT FILE. CONTINUE BY TYPING COMMANDS'
is output. The user must complete the edit by typing commands from the teletype.

¶*Double editing.* It is not possible to do double editing using POPEDIT.

¶*Global variables.* All global variables used in the program are prefixed by POPED except for the function POPGOBBLE.

¶*Space used.* The compiled program LIB POPEDIT requires approximately 2 blocks of store.

The program does not use much working space.

¶*Examples of use.* In the following examples it is assumed that the file to be edited (the source file) is represented by the character repeater INFILE, various output files are represented by the character repeaters OUTLP, OUTPT, OUTDISC and so on, and the file of edit commands is either the character repeater CHARIN, if the editing is being done on-line, or EDITCOMMANDS.

These character repeaters are all created in the standard way: for example, for papertape

```
POPMESS([PTIN INPUTFILE]) –> INFILE;
POPMESS([PTIN EDITS]) –> EDITCOMMANDS;
POPMESS([PTOUT EDIT FILE]) –> OUTPT;
```

for disc

```
DISCIN(TR, N) –> INFILE;
DISCOUT(TR, N) –> OUTDISC;
```

where TR and N are the required disc track and sector numbers.
for lineprinter

```
POPMESS([LP80 EDITED FILE]) –> OUTLP;
```

Online editing. (1) To edit, from the teletype, a file, and to compile the edited file without outputting it to any device, type:

```
COMPILE(POPEDIT(INFILE, CHARIN, NIL));
```

This should be followed by the edit commands.
(2) To edit, from the teletype, a file, and to compile the edited file, and copying the new file to the disc, type:

```
COMPILE(POPEDIT(INFILE, CHARIN, [%OUTDISC%]));
```

This should again be followed by the edit commands.

(3) To edit, from the teletype, a file to compile the edited file, and to copy the new file onto both the disk and the line-printer, type:

```
COMPILE(POPEDIT(INFILE, CHARIN, [%OUTDISC, OUTLP%]));
```

Again, follow this with the edit commands.
(4) If simultaneous compilation, as above, is not required, COMPILE, in all cases, should be replaced by POPGOBBLE.

For example, to edit a file without compiling it, but producing a new file on disc, a paper-tape copy, and a listing to the line-printer, the user would type:

```
POPGOBBLE(POPEDIT(INFILE, CHARIN, [%DISCOUT, OUTLP,
OUTPT%]));
```

followed by the edit commands.

Offline editing. If the user wishes to use an offline file of edit commands, for example a paper-tape which he had typed up previously, in all cases in the above examples, the character repeater for the teletype, CHARIN, would be replaced by that for the file of edit commands. For example,

```
COMPILE(POPEDIT(INFILE, EDITCOMMANDS, [%OUTLP%]));
```

to edit and compile simultaneously, the file INFILE, from the file of edit commands EDITCOMMANDS, producing a line-printer listing of the new file.

If the edit is successful then the message:
'EDIT FINISHED.   OUTPUT FILES CLOSED'
will be output, otherwise an error message as described, will be given.

Complete examples of the use of POPEDIT. The following file is assumed to be on track TR sector N of the disc:

```
FUNCTION SIGMA L;
   IF L.NULL THEN 0 ELSE L.HD + SIGMA(TL(L))
   CLOSE
END;

FUNCTION FACT N; N*FACT(N—1);
END;

"A" —> A;
1, 2, 3 —> L;
FACT(A) =>
SIGMA(LIST) =>
```

It is required to edit this file, compiling it, and copying it back to disc track TR1, sector N1, producing a paper-tape copy.

The required input on the teletype is given below, assuming editing is to be done online.

```
: VARS AA BB CC;                              Comments
: DISCIN(TR, N) —> AA;
: DISCOUT(TR1, N1) —> BB;
: POPMESS([PTOUT EDITED FILE]) —> CC;         Giving the paper tape
                                              file the name EDITED
                                              FILE.

: COMPILE(POPEDIT(AA, CHARIN,                 Calling POPEDIT.
  [%BB, CC%]));

'READY:  TYPE EDIT COMMANDS'                  Output by POPEDIT to
                                              inform user that he
                                              may start typing in
                                              commands.

: FL END
: FE
: ON
: FC ;

FUNCTION FACT N;                              Output due to the ON
: IL    IF N = 0 THEN 1 ELSE                  command followed by
                                              the next edit command
                                              after the ":"
```

```
IF N = 0 THEN 1 ELSE : FC)
N*FACT(N-1) : IS CLOSE
CLOSE : OFF                                       Switch off teletype
: FC E;                                           output
: IB

VARS A LIST;
   2—> A;  [% 1, 2↑2, 3 %] —> LIST;

↑                                                 Inserted block
: DL 1                                            terminated with ↑
: DE
: SH
** 2,                                             Output due to the two
                                                  print statements in
** 8.0,                                           the source file

'EDIT FINISHED.    OUTPUT FILES CLOSED'
:
```

The new file produced by the above edit commands on the given file is given below. This file will have been punched to paper-tape, and also written onto the disc:

```
FUNCTION SIGMA L;
   IF L.NULL THEN 0 ELSE L.HD + SIGMA(TL(L))
   CLOSE
END;

FUNCTION FACT N;
   IF N=0 THEN 1 ELSE N*FACT(N—1) CLOSE;
END;

VARS A LIST;
   2—> A;  [% 1, 2↑2, 3%] —> LIST;

FACT(A) =>
SIGMA(LIST) =>
```

```
[POPEDIT]

VARS POPEDCU POPEDERR POPEDT POPEDED POPEDED1 POPEDRD POPEDREP POPEDC
     POPEDOCU POPEDL POPEDIN POPED2 POPED4 POPED5 POPED6
     POPEDSTR POPEDON POPEDBK POPEDF1 POPEDF2 POPEDF POPEDPTR POPEDCUC;


FUNCTION POPGOBBLE POPED;
 L0: IF .POPED=TERMIN THEN EXIT; GOTO L0
END;


FUNCTION POPED1;
 L0:
  .POPEDCU->POPEDCUC;
  IF TERMIN=POPEDCUC THEN
    1.NL; 'END OF SOURCE FILE. OUTPUT FILES CLOSED`.PR; 1.NL;
    IF NOT(POPEDED1=CHARIN) THEN POPEDED1.POPGOBBLE CLOSE
  EXIT;
  IF POPEDCUC=POPEDF THEN
    IF POPEDPTR THEN .POPEDT
    ELSEIF POPEDF1 THEN .POPED2 ELSE POPED2->POPEDREP CLOSE;
  ELSEIF POPEDF2 AND NOT(POPEDCUC=16) THEN .POPEDT CLOSE;
  IF POPEDF1 THEN GOTO L0 CLOSE
END;


FUNCTION POPEDILF; POPEDIN->POPEDCU; POPEDIN->POPEDOCU; 17 END;


FUNCTION POPED2;
VARS U V;
 L0:
  .POPEDED->U; IF U=17 OR U=16 THEN GOTO L0 CLOSE;
  .POPEDED->V; POPEDOCU->POPEDCU; 0->POPEDF1;
  IF U=41 THEN .POPED4 EXIT;
  IF U=51 THEN .POPED5 EXIT;
  IF U=47 THEN .POPED6; GOTO L0 CLOSE;
  0->POPEDF2;
  IF U=38 THEN 0 ELSEIF U=36 THEN 1 ELSE U.POPEDERR EXIT -> POPEDF1;
  IF V=44 THEN .POPEDRD->POPEDSTR; POPEDL->POPEDT; 17->POPEDF
  ELSEIF V=35 THEN .POPEDRD.NEXT->POPEDSTR->POPEDF; POPEDC->POPEDT
  ELSEIF V=37 THEN POPED2->POPEDT; 1->POPEDSTR; POPEDILF->POPEDOCU; 17->POPEDF
  ELSE V.POPEDERR EXIT;
  POPEDSTR->POPEDPTR; POPED1->POPEDREP; .POPED1
END;

FUNCTION POPED4;
  1->POPEDF2;
  IF V=34 THEN 1->POPEDBK; .POPEDRD->POPEDPTR
  ELSEIF V=44 THEN 17::POPEDRD()->POPEDPTR
  ELSEIF V=51 THEN .POPEDRD->POPEDPTR
  ELSE V.POPEDERR EXIT;
  LAMBDA;
    IF POPEDPTR THEN POPEDPTR.HD->POPEDCUC; POPEDPTR.TL->POPEDPTR
    ELSE POPED1->POPEDREP; .POPED2 CLOSE
  END ->POPEDREP; .POPEDREP
END;



FUNCTION POPEDSH;
  .POPEDCU->POPEDCUC;
  IF TERMIN=POPEDCUC THEN
  1.NL; 'EDIT FINISHED. OUTPUT FILES CLOSED`.PR; 1.NL CLOSE
END;


FUNCTION POPED5;
  IF V=40 THEN POPEDSH->POPEDREP; .POPEDREP EXIT;
  V.POPEDERR
END;


FUNCTION POPED6;
  IF V=46 THEN 1->POPEDON EXIT;
  IF V=38 THEN .POPEDED->V; IF V=38 THEN 0->POPEDON EXIT CLOSE;
  V.POPEDERR
END;
```

```
FUNCTION POPEDRD;
VARS U V W;
  NIL::0->U; U->V;
  .POPEDED->W;
  IF W=16 THEN L0: .POPEDED->W CLOSE;
 L1:
  IF W=17 THEN
    IF POPEDBK THEN
      .POPEDED->W; IF W=62 THEN 0->POPEDBK; U.TL EXIT;
      17::0->V.TL; V.TL->V; GOTO L1
    ELSE U.TL EXIT
  CLOSE;
  IF W=16 AND POPEDF2.NOT THEN GOTO L0 CLOSE;
  W::0->V.TL; V.TL->V; GOTO L0
END;


FUNCTION POPEDC; POPEDPTR.DEST->POPEDPTR->POPEDF END;


FUNCTION POPEDL;
  IF POPEDCUC=POPEDF THEN
   L0:
    POPEDPTR.HD->POPEDF; POPEDPTR.TL->POPEDPTR; 1->POPEDF2
  ELSE
    POPEDSTR->POPEDPTR; IF POPEDCUC=17 THEN GOTO L0 CLOSE;
    17->POPEDF; 0->POPEDF2
  CLOSE
END;


FUNCTION POPEDERR;
  CHAROUT->CUCHAROUT;
  1.NL; 'POPEDERR    CULPRIT  `.PR; .CHAROUT; 1.NL;
  IF POPEDED1=CHARIN THEN 'TRYAGAIN`
  ELSE 'CONTINUE BY TYPING EDIT COMMANDS`; POPEDED1.POPGOBBLE; CHARIN->POPEDED1
  CLOSE.PR; 1.NL; .POPED2
END;


FUNCTION POPEDED;
VARS U;
  .POPEDED1->U;
  IF TERMIN=U THEN
    1.NL; 'END OF EDIT FILE. CONTINUE BY TYPING COMMANDS`.PR; 1.NL;
    CHARIN->POPEDED1; .CHARIN;
  EXIT;
  U
END;


FUNCTION POPEDSTART;
  IF POPEDED1=CHARIN THEN 1.NL; 'READY: TYPE EDIT COMMANDS`.PR; 1.NL CLOSE;
  .POPED2
END;


FUNCTION POPEDIT POPEDF;
  ->POPEDED1 ->POPEDIN; 0->POPEDON; 0->POPEDBK; POPEDIN->POPEDOCU;
  POPEDSTART->POPEDREP;
  POPVAL([LAMBDA; .POPEDREP; IF POPEDON THEN POPEDCUC.CUCHAROUT CLOSE;]<>
  [%APPLIST(POPEDF,LAMBDA X; "POPEDCUC",",",X,".","APPLY",";" END),
  "POPEDCUC","END",";","GOON" %]);
END;

1.NL; 'POPEDIT READY FOR USE`.PR; 2.NL;
```

Program name. LIB POPSTATS
Source. R. M. Burstall, S. Weir, D. Michie, DMIP; *Date of issue.* December 1968.

¶*Description.* This program provides a conversational, online, statistical package which can be used by people with little or no knowledge of computers.

¶*How to use program.* The program should be compiled by typing:
COMPILE(LIBRARY([LIB POPSTATS]));
The background to this program is fully described in memoranda MIP-R-38/39.

To enter the program the user should type:
. POPSTATS;
to which the computer will reply with the message:
[TYPE POPSTATS COMMAND]:
If the user is in any doubt as to what should be given as the reply to any request of the above form, then the word HELP should be typed.

The program allows the user to input data from both the teletype and from paper-tape, to edit the data, to print the data out again, and to apply various statistical routines to it.

¶*Warning.* LIB POPSTATS is a large program, 32 blocks of store being needed by the user when running it.

REFERENCES

Burstall, R. M. (1968) The helpful civil servant, a conversational control routine. *Research Memorandum MIP-R-38.* Edinburgh: Department of Machine Intelligence and Perception.

Michie, D. & Weir, S. (1968) Application of Burstall's control routine to conversational statistics. *Research Memorandum MIP-R-39*. Edinburgh: Department of Machine Intelligence and Perception.

```
[POPSTATS]


VARS MATRIX TABLE CHARCOUNT LINELENGTH SIGNOFF RANSEED PRTABLE
R1 NUM  SUBTABF REP FF CRUNCH; 0->PRTABLE;


NL(1);  PR('POPSTATS NOW COMPILING');

0 -> CHARCOUNT;  64 -> LINELENGTH;  NIL -> MATRIX;  NIL -> TABLE;
['TYPE END WHEN FINISHED'] -> SIGNOFF;


FUNCTION FORM X;
    LOGAND(X,8:77)->X;
    IF CHARCOUNT>LINELENGTH AND X=16 THEN 17->X CLOSE;
    IF X=23 OR X=32 THEN EXIT
    IF X=17 THEN 0->CHARCOUNT
     ELSE CHARCOUNT + 1 ->CHARCOUNT CLOSE;
    CHAROUT(X)
END

10->RANSEED;


FUNCTION RANDOM;
  (125*RANSEED+1)//16384; .ERASE; ->RANSEED;
  RANSEED/16384;
END



FUNCTION QQAVERAGE LIST;
VARS N;
INTOF(.RANDOM*LENGTH(LIST))->N;
LOOP:IF N=0 THEN LIST.HD ELSE LIST.TL->LIST;N-1->N; GOTO LOOP CLOSE
END



FUNCTION QUICKSORT LIST;
VARS Y Z Q QQV QQW QQS;
0;
L2:IF NULL(LIST) OR NULL(TL(LIST)) THEN GOTO SPLIT CLOSE;
NIL->QQS;NIL->Y;NIL->Z;
QQAVERAGE(LIST)->QQW;
L1:HD(LIST)->QQV;TL(LIST)->LIST;
IF QQW>QQV THEN QQV::QQS->QQS
  ELSEIF QQW<QQV THEN QQV::Z->Z
  ELSE QQV::Y->Y
CLOSE;
IF NULL(LIST) THEN Z;Y;1;QQS->LIST;GOTO L2 ELSE GOTO L1 CLOSE;
SPLIT: ->Q;IF Q=0 THEN LIST EXIT
->Y;
IF Q=1 THEN ->Z;LIST<>Y;2;Z->LIST;GOTO L2 CLOSE;
Y<>LIST->LIST;
GOTO SPLIT
END



FUNCTION MOSTEST LIST COMPARE;
  VARS NOWVAL NEXTVAL;
  LIST.HD->NOWVAL;
  LOOP: LIST.TL -> LIST;
  IF LIST.NULL THEN NOWVAL EXIT;
  LIST.HD -> NEXTVAL;
  IF COMPARE(NEXTVAL,NOWVAL) THEN NEXTVAL -> NOWVAL CLOSE;
  GOTO LOOP
END


FUNCTION GET LIST NTH;
  LOOP: IF LIST.NULL OR NOT(NTH.ISINTEGER) THEN UNDEF EXIT;
  IF NTH = 1 THEN LIST.HD EXIT;
  LIST.TL -> LIST;
  NTH - 1 -> NTH;
  GOTO LOOP
END
```

```
FUNCTION MEMBER OBJECT LIST;
  LOOP: IF LIST.NULL THEN FALSE EXIT;
  IF OBJECT = LIST.HD THEN TRUE EXIT;
  LIST.TL ->LIST;
  GOTO LOOP
END

VARS OPERATION 2 <->;

FUNCTION <-> X Y;
  IF NULL(X) THEN Y EXIT
  X;
 LO:
  IF NULL(BACK(X)) THEN Y->BACK(X) EXIT
  BACK(X)->X;  GOTO LO
END;



FUNCTION ASSOC X Y LIST;
  VARS HOLD;
  LIST -> HOLD;
  LOOP: IF LIST.NULL THEN (X::Y)::HOLD EXIT;
  IF LIST.HD.FRONT = X THEN Y -> LIST.HD.BACK;
  HOLD EXIT;
  LIST.TL -> LIST;
  GOTO LOOP
END


FUNCTION ASSOCVAL X LIST;
  LOOP: IF LIST.ATOM THEN UNDEF EXIT;
  IF X = LIST.HD.FRONT THEN LIST.HD.BACK EXIT;
  LIST.TL -> LIST;
  GOTO LOOP
END


FUNCTION ASSOCGET LIST IDENT;
IF IDENT.ISINTEGER.NOT THEN NUM(IDENT,LIST)->IDENT CLOSE;
  GET(LIST,IDENT);
END


FUNCTION ASSOCPR LIST;
  LOOP: IF LIST.NULL THEN NL(1) EXIT;
  NL(1);
  PR(LIST.HD.FRONT);
  SP(1);
  PR(LIST.HD.BACK);
  LIST.TL -> LIST;
  GOTO LOOP
END


FUNCTION REASSOC X Y LIST;
  LOOP: IF LIST.NULL THEN EXIT;
  IF LIST.HD.FRONT = Y THEN X -> LIST.HD.FRONT EXIT;
  LIST.TL -> LIST;
  GOTO LOOP
END



FUNCTION INPUTCHAT REQUEST CLUB ELIGIBLE WARNING TYPE;
VARS OBJECT COUNT LIST M LASTOB ERRFUN INPUT;
  FUNCTION ERRFUN X N;
    IF N=33 THEN CRUNCH('TOO LARGE A NUMBER`)
    ELSE CRUNCH('ILLEGAL NAME USED`) CLOSE
  END;
  0->COUNT; 0->M; NIL->LIST; 0->LASTOB;
LOOP: CHARIN.INCHARITEM->INPUT;
  IF COUNT>2 THEN CRUNCH('STILL`::WARNING) CLOSE;
  IF COUNT>0 THEN NL(1); PR(WARNING); NL(1); CLOSE;
  COUNT+1->COUNT; NL(1);
  PR('TYPE`::REQUEST);SP(1);
LISTLOOP: INPUT()->OBJECT;
  IF OBJECT="-" OR OBJECT="+" THEN OBJECT->LASTOB; GOTO LISTLOOP CLOSE;
IF LASTOB="-" AND OBJECT.ISNUMBER THEN -OBJECT->OBJECT;CLOSE;0->LASTOB;
```

```
 IF NOT(MEMBER(OBJECT,CLUB)) AND NOT(OBJECT.ELIGIBLE) THEN
   IF TYPE="THING" THEN
     GOTO LOOP
   ELSE
     M+1->M;
     IF M>2 THEN CRUNCH('STILL`::(WARNING<>['START AGAIN`])) CLOSE;
     NL(1);PR('ITEM`::(LENGTH(LIST)+1::(WARNING<>
         [', IT AND ALL SUBSEQUENT HAVE BEEN IGNORED,  CARRYON`])));NL(1);
 CHARIN.INCHARITEM->INPUT;
    GOTO LISTLOOP
   CLOSE;
 CLOSE;
 IF TYPE="THING" THEN OBJECT EXIT
 IF NOT(OBJECT="END") THEN LIST<->[%OBJECT%]->LIST; GOTO LISTLOOP CLOSE;
LIST
END


FUNCTION DUMMY X;
  FALSE
END
FUNCTION GOOD X;
  TRUE
END


FUNCTION ISNAME WORD;
  IF ASSOCVAL(WORD,MATRIX) = UNDEF THEN FALSE ELSE TRUE
 CLOSE;
END


FUNCTION ELIGNEW WORD;
  IF NOT(WORD.ISWORD)  OR WORD.ISNAME THEN FALSE  ELSE TRUE CLOSE;
END



FUNCTION INCOL REQUEST;
  VARS NAME NUMBERS C; 0->C;
LOOP:
  INPUTCHAT(REQUEST,[END],ISWORD,['WORD NOT NUMBER PLEASE`],"THING") -> NAME;
  IF NAME.ISNAME THEN C+1->C;
    IF C<3 THEN NL(1);  PR(['NAME USED ALREADY`]); NL(1); GOTO LOOP
    ELSE CRUNCH (['NAME USED ALREADY, START AGAIN`]) CLOSE; CLOSE;
  IF NAME = "END" THEN 1.NL; FALSE EXIT;
  INPUTCHAT('NUMBERS,`::SIGNOFF,[END],
  ISNUMBER,['NOT A NUMBER`],"LIST") -> NUMBERS;
  IF NUMBERS.NULL THEN NAME::NIL ELSE NAME::NUMBERS CLOSE;
END




FUNCTION INBOUNDS COLNO;
  IF COLNO.ISINTEGER  AND COLNO>0 AND COLNO=<LENGTH(MATRIX) THEN TRUE ELSE FALSE
END


FUNCTION ISCOL COLIDENT;
  IF COLIDENT.ISNAME OR COLIDENT.INBOUNDS THEN TRUE ELSE FALSE CLOSE;
END


FUNCTION FETCHCOL REQUEST;
  VARS COLIDENT;
  INPUTCHAT(REQUEST,NIL,ISCOL,['NO SUCH COLUMN`],"THING") -> COLIDENT;
  ASSOCGET(MATRIX,COLIDENT)
END



FUNCTION INPUTFUN REQUEST;
VARS INT LIST ERRFUN;
  FUNCTION ERRFUN X N;
    IF N=33 THEN CRUNCH(['NUMBER TOO LARGE`]) EXIT;
    CRUNCH(['NOT AN EXPRESSION`]);
  END;
  NIL->LIST;
LOOP:
  PR('TYPE`::REQUEST); SP(1);
LISTLOOP:
  ITEMREAD()->INT;
  IF INT="END" THEN 1.NL;
  PR('TRANSFORMATION FUNCTION FOR CHECKING:`::LIST);
  POPVAL([LAMBDA X;]<>LIST<>[END->FF;  GOON]); EXIT
  LIST<->[%INT%]->LIST; GOTO LISTLOOP
END


NL(1);  PR('STILL COMPILING`);


FUNCTION SPLIT X N;
  VARS FIRST; NIL->FIRST;
LOOP: IF N=0 THEN FIRST,X EXIT
     FIRST<-> [%X.HD%]->FIRST;
     X.TL->X; N-1->N; GOTO LOOP
END


FUNCTION ITEMLENGTH ITEM;
VARS COUNT RCOUNT M CUCHAROUT;
  FUNCTION CUCHAROUT X;
    IF X=14 THEN TRUE->M CLOSE;
    IF M THEN RCOUNT+1->RCOUNT ELSE COUNT+1->COUNT CLOSE
  END;
  0 -> COUNT;  0 -> RCOUNT;  0 -> M;
  PR(ITEM);  RCOUNT, COUNT;
END


FUNCTION MAXDEC LIST;
VARS Y RCOUNT WPB;
  0->Y;  0->RCOUNT;
  APPLIST(LIST,
           LAMBDA X; X.ITEMLENGTH->WPB->Y; IF Y>RCOUNT THEN Y->RCOUNT CLOSE END);
  RCOUNT;
END;


FUNCTION DROP X LIST;
VARS NEWLIST NEXTVAL;
  NIL -> NEWLIST;
 LOOP:
  IF LIST.NULL THEN NEWLIST EXIT;
  LIST.HD -> NEXTVAL;
  IF NEXTVAL = X THEN NEWLIST<->LIST.TL EXIT;
  NEWLIST<->[%NEXTVAL%] -> NEWLIST;
  LIST.TL -> LIST;
  GOTO LOOP
END


FUNCTION NUM WORD LIST;
VARS N;
  0 -> N;
 LOOP:
  IF LIST.NULL THEN UNDEF EXIT N+1->N;
  IF WORD=LIST.HD.HD  THEN N EXIT
  LIST.TL->LIST; GOTO LOOP
END


FUNCTION COLPRINT LIST;
VARS LONG LONGEST SECOND N D E F G H;
IF LIST.NULL THEN NL(1);  PR (['NO DATA`]); NL(1);  EXIT;
  1.NL;0->LONGEST; 0->H; 0->D;
APPLIST(LIST,LAMBDA X;
              LENGTH(X)->LONG; IF LONG>LONGEST THEN LONG->LONGEST CLOSE;
              MAXDEC(X)->LONG; IF LONG>D THEN LONG->D CLOSE;
            END);
 LOOP:0->N;
  NIL -> SECOND;
  IF LENGTH(LIST)>LINELENGTH/12.5 THEN
    SPLIT(LIST,INTOF(LINELENGTH/12.5)) ->SECOND ->LIST;
  CLOSE;
  APPLIST(LIST.HD,
          LAMBDA X;
            X.ITEMLENGTH ->E->F;
            IF E>H THEN E->H CLOSE
          END);
 LISTLOOP:
  N+1 -> N;
  IF NOT(N>LONGEST) THEN
    NL(1);  H-9->G;
    APPLIST(LIST,
            LAMBDA X;
              IF X.NULL THEN 9+D+G->G ELSE
                X.HD.ITEMLENGTH->E->F;
                IF X.HD.ISWORD THEN
                  IF MAXDEC(X)>2 THEN
                    E-2->E;  F+2->F;
                  ELSEIF MAXDEC(X)=2 THEN
                    E-1->E;  F+1->F;
                  CLOSE
                CLOSE;
                SP(9-E+G);  PR(X.HD);  D-F->G;
              CLOSE
            END );
    MAPLIST(LIST, LAMBDA X; IF X.NULL THEN NIL ELSE X.TL CLOSE END)
      -> LIST;
    GOTO LISTLOOP;
  ELSEIF SECOND.NULL.NOT THEN
    SECOND->LIST;  2.NL; GOTO LOOP;
  CLOSE;
  2.NL;
END


FUNCTION POWERSUM LIST POWER;
  VARS ACCUM;
  0->ACCUM;
  LOOP:IF LIST.NULL THEN ACCUM EXIT;
  ACCUM+LIST.HD↑POWER->ACCUM;
  LIST.TL->LIST;
  GOTO LOOP
END


FUNCTION MEAN LIST;
  POWERSUM(LIST,1)/LENGTH(LIST);
END


FUNCTION MEDIAN LIST;
VARS MID;
   QUICKSORT(LIST)->LIST;
  IF(LENGTH(LIST)//2->MID)=1 THEN GET(LIST,MID +1) EXIT
  (GET(LIST,MID) +GET(LIST,MID +1))/2
END


FUNCTION SUMSQDEV LIST;
  POWERSUM(LIST,2)-(POWERSUM(LIST,1)↑2)/LENGTH(LIST)
END


FUNCTION STANDEV LIST;
  SQRT(SUMSQDEV(LIST)/(LENGTH(LIST)-1))
END


FUNCTION TTEST LIST MU;
  VARS N;
  LENGTH(LIST)->N;
  [%(MEAN(LIST)-MU)*SQRT(N)/STANDEV(LIST),N-1%]
END
```

```
FUNCTION PRODSUM LIST1 LIST2;
  VARS ACCUM;
  0->ACCUM;
  LOOP:IF LIST1.NULL THEN ACCUM EXIT;
  ACCUM+LIST1.HD*LIST2.HD->ACCUM;
  LIST1.TL->LIST1;
  LIST2.TL->LIST2;
  GOTO LOOP
END


FUNCTION CORREL LIST1 LIST2;
  (PRODSUM(LIST1,LIST2)-POWERSUM(LIST1,1) *POWERSUM(LIST2,1)/LENGTH(LIST1))
    /(SQRT(SUMSQDEV(LIST1))*SQRT(SUMSQDEV(LIST2)))
END


FUNCTION SCAN LIST LOWBOUND LIMIT;
  LOOP:IF LIST.NULL THEN EXIT;
  IF LIST.HD>=LOWBOUND AND LIST.HD<LIMIT THEN PR("X") CLOSE;
  LIST.TL->LIST;
  GOTO LOOP
END


FUNCTION HISTO LIST LOWBOUND STEP LIMIT;
  LOOP:IF LOWBOUND>LIMIT THEN NL(1) EXIT;
  NL(1);
  PR([%LOWBOUND,"-"%]);SP(15-CHARCOUNT);
  SCAN(LIST,LOWBOUND,LOWBOUND+STEP);
  LOWBOUND+STEP->LOWBOUND;
  GOTO LOOP
END


FUNCTION LFTABLE N;
  GET([0.0000 0.6931 1.792 3.178 4.788 6.579 8.525 10.60 12.80 15.10 17.50
    19.99 22.55 25.19 27.90 30.67 33.50 36.40 39.34 42.34],N);
END



FUNCTION STIRLING N;
  (0.5*LOG(6.284))+((N+0.5)*LOG(N))-(N-1/(12*N))
END;


FUNCTION LOGFACT N;
  IF N = 0 THEN 0.0000
  ELSEIF N < 21   THEN  LFTABLE(N)
   ELSEIF N > 40 THEN STIRLING(N)
   ELSE LOG(N) + LOGFACT(N -1)
  CLOSE;
END


FUNCTION FISHER  L;
VARS A B C D ACCUM SORTLIST DEC1 N; FALSE->DEC1; 0->ACCUM;
     L.HD.HD->A; L.HD.TL.HD ->B; L.TL.HD.HD->C; L.TL.HD.TL.HD->D;
QUICKSORT(A::(B::(C::[%D%])))->SORTLIST;SORTLIST.HD->N;

IF N=A OR N=D  THEN TRUE->DEC1;CLOSE;
LOOP:ACCUM+
     EXP(LOGFACT(A+C)+LOGFACT(B+D)+LOGFACT(A+B) +LOGFACT(C+D) -LOGFACT(A+B+C+D)
     -LOGFACT(A) -LOGFACT(B) -LOGFACT(C) -LOGFACT(D))->ACCUM;
IF N=0 THEN'EXACT TEST GIVES PROBABILITY OF'::( ACCUM ::[',FOR A ONE-TAILED TEST']
N-1->N;
IF DEC1
THEN A-1->A;D-1->D;B+1->B;C+1->C;
ELSE    A+1->A; D+1->D;B-1->B;C-1->C;
CLOSE
GOTO LOOP
END
```

```
FUNCTION MATRIXSUM L P;
  VARS ACCUM;  0-> ACCUM;
  L1: IF NOT (L.NULL)
   THEN  ACCUM + POWERSUM(L.HD,P) -> ACCUM;
         L.TL-> L;  GOTO L1   CLOSE;
     ACCUM
END

FUNCTION ROWSUM ROWNO L;
  VARS ACCUM;  0-> ACCUM;
  L1: IF NOT (L.NULL)
      THEN ACCUM + GET(L.HD,ROWNO) -> ACCUM;
      L.TL -> L; GOTO L1  CLOSE;
    ACCUM
END


FUNCTION EEVAL X ROWNO L;
  (ROWSUM(ROWNO,L)*POWERSUM(X,1)/MATRIXSUM(L,1))
END


FUNCTION DF N;
  IF N=1 THEN 1 ELSE N-1 CLOSE
END


FUNCTION BINOM LIST;
VARS MINVAL SIGNIF  N; 0->SIGNIF;
MOSTEST(LIST,NONOP <)->MINVAL; POWERSUM(LIST,1)->N;
IF N>=8 AND MINVAL=<1 THEN TRUE->SIGNIF;
ELSEIF MINVAL=0 AND N>=5 AND N<8 THEN TRUE->SIGNIF;  CLOSE;
IF SIGNIF THEN ['SIGNIFICANT AT 5 PERCENT LEVEL`]
ELSE ['NOT SIGNIFICANT AT 5 PERCENT LEVEL`]; CLOSE
END
FUNCTION YATES OBS E ACC;
  IF OBS<E THEN ACC +((OBS-E+0.5 )↑2/E)
  ELSEIF OBS=INTOF(E) THEN ACC+ (OBS-E)↑2/E
  ELSE ACC+((OBS-E-0.5)↑2/E)
  CLOSE
END


FUNCTION CHISQ L;
VARS X OBS E ROWNO COLNO ACCUM LOWEVAL COMBIN  XX LL; 0->COMBIN;0->LOWEVAL;
0->SIGNIF;  0->COLNO;  0->ACCUM; L->LL;
 L1:
  L.HD->X; X->XX; COLNO+1->COLNO;  0->ROWNO;
 L2:
  X.HD->OBS; ROWNO+1->ROWNO;
  IF LENGTH(LL) =1 THEN MEAN(LL.HD) -> E; ELSE  EEVAL(XX,ROWNO,LL)->E   CLOSE;
IF LL.LENGTH=2 AND E<5 THEN IF LL.HD.LENGTH=2
                         THEN FISHER(LL); EXIT;TRUE->COMBIN;CLOSE;
IF LL.LENGTH=1 AND E<5 THEN IF LL.HD.LENGTH=2
                            THEN BINOM(LL.HD);EXIT
 TRUE->COMBIN; CLOSE;
L3:
IF COMBIN THEN ['EXPECTED VALUES TOO LOW,TRY COMBINING CATEGORIES`] EXIT;
YATES(OBS,E,ACCUM)->ACCUM;IF E<5 THEN LOWEVAL+1->LOWEVAL;CLOSE;
  X.TL->X;
  IF NOT(X.NULL) THEN GOTO L2 CLOSE;
  L.TL->L;
  IF NOT (L.NULL) THEN GOTO L1 CLOSE;
IF (LOWEVAL/(LENGTH(LL.HD)*LENGTH(LL)))>0,2 THEN TRUE->COMBIN;GOTO L3;CLOSE;
    [% ACCUM,,DF(ROWNO)*DF(COLNO) %]
END


NL(1); PR('STILL COMPILING`);


FUNCTION GENSTATF;
VARS DATALST;
  TL(FETCHCOL(['COLUMN NAME OR NUMBER`])) -> DATALST;
  IF NOT(LENGTH(DATALST)>1) THEN CRUNCH(['TOO FEW NUMBERS IN COLUMN`]) CLOSE;
  NL(1);
  PR('MEDIAN =`::(MEDIAN(DATALST)::('  MEAN =`::(MEAN(DATALST)
  ::('  STAND DEVN =`::[%STANDEV(DATALST)%])))));
  NL(1)
END
```

```
FUNCTION TTESTF;
VARS TLIST DATALST;
  TL(FETCHCOL(['COLUMN NAME OR NUMBER`]))->DATALST;
  IF NOT(LENGTH(DATALST)>1) THEN CRUNCH(['TOO FEW NUMBERS IN COLUMN`]) CLOSE;
  TTEST(DATALST,INPUTCHAT(['ASSUMED MEAN`],NIL,ISNUMBER,
                          ['NOT NUMBER`],"THING")) -> TLIST;
  NL(1);
  PR('T IS`::(TLIST.HD::(',  DF =`::TLIST.TL)));
  NL(1)
END



FUNCTION CORRELF;
  VARS IDENTLIST DATACOL1 DATACOL2 COUNT;
  0 -> COUNT;
  LOOP:
  INPUTCHAT('COLUMN NAMES OR NUMBERS;`::SIGNOFF,
            [END],ISCOL,['NO SUCH COLUMN`],"LIST") -> IDENTLIST;
  COUNT + 1 -> COUNT;
  IF NOT(LENGTH(IDENTLIST) = 2) THEN
    IF COUNT < 3 THEN
      NL(1);  PR(['TWO COLUMNS PLEASE`]); NL(1); GOTO LOOP
    ELSE
      CRUNCH(['STILL NO GOOD`])
    EXIT
  CLOSE;
  TL(ASSOCGET(MATRIX,IDENTLIST.HD)) -> DATACOL1;
  TL(ASSOCGET(MATRIX,IDENTLIST.TL.HD)) -> DATACOL2;
  IF NOT(LENGTH(DATACOL1) = LENGTH(DATACOL2)) THEN
    IF COUNT <3 THEN
      NL(1); PR(['UNEQUAL COLUMN LENGTHS`]); NL(1); GOTO LOOP
    ELSE
      CRUNCH(['STILL NO GOOD`])
    EXIT
  CLOSE;
IF LENGTH(DATACOL1)<2 THEN CRUNCH ((['TOO FEW ROWS IN COLUMNS`])CLOSE;
  NL(1);
  PR('CORREL =`::[%CORREL(DATACOL1,DATACOL2)%]);
  NL(1)
END



FUNCTION HISTOF;
  VARS DATACOL LOWBOUND STEP LIMIT MINVAL MAXVAL;
  TL(FETCHCOL(['COLUMN NAME`]))->DATACOL;
IF LENGTH(DATACOL)<2 THEN CRUNCH(['TOO FEW ROWS IN COLUMN`])CLOSE;

  MOSTEST(DATACOL,NONOP<)->MINVAL;
  MOSTEST(DATACOL,NONOP>)->MAXVAL;
  INPUTCHAT(['LOWER BOUND`],NIL,LAMBDA X; IF X.ISNUMBER AND X=<MINVAL
      THEN TRUE ELSE FALSE CLOSE; END,['NOT A LOWER BOUND`],"THING")->LOWBOUND;
  INPUTCHAT(['UPPER BOUND`],NIL,LAMBDA X; IF X.ISNUMBER AND X>=MAXVAL
      THEN TRUE ELSE FALSE CLOSE; END,['NOT AN UPPER BOUND`],"THING")->LIMIT;
  INPUTCHAT(['INTERVAL`],NIL,LAMBDA X; IF X.ISNUMBER AND
      X>(MAXVAL-MINVAL)/15 AND X<(MAXVAL-MINVAL)/3 THEN TRUE ELSE
      FALSE CLOSE;  END,['NOT A SUITABLE INTERVAL`],"THING")->STEP;
  HISTO(DATACOL,LOWBOUND,STEP,LIMIT)
END



FUNCTION RESCALEF;
VARS NEWNAME OLDCOL FUN;
  INPUTCHAT(['NAME FOR COLUMN OF TRANSFORMED VALUES`],NIL,ELIGNEW,
      ['NAME USED ALREADY`],"THING")->NEWNAME;
  FETCHCOL(['NAME OR NUMBER OF COLUMN TO BE RESCALED`])->OLDCOL;
  NL(1);
  INPUTFUN('EXPRESSION IN ONE VARIABLE, X,  THUS:
    X*X, LOG(X+1), ETC.
`::SIGNOFF);
MATRIX<->[% NEWNAME::MAPLIST(OLDCOL.TL,FF)%]->MATRIX;
  NL(2);
  PR('RESCALED VERSION FOR CHECKING:

`::ASSOCGET(MATRIX,NEWNAME));NL(2);
END
```

```
FUNCTION CONTINGF;
  VARS CHITABLE CHILIST COUNT Y C;  0->COUNT; NIL->TABLE;
 LOOP:
       IF COUNT=3 THEN CRUNCH(['STILL UNEQUAL COLUMNS']) EXIT
      IF COUNT THEN 1.NL; PR(['UNEQUAL COLUMN LENGTHS']); 1.NL; CLOSE;
       COUNT+1->COUNT; TRUE->C;
       SUBTABF(); IF TABLE.NULL THEN CRUNCH(['NO DATA']) EXIT
       MAPLIST(TABLE, LAMBDA X; X.TL END)->CHITABLE;
       LENGTH(CHITABLE.HD)->Y;
       APPLIST(CHITABLE, LAMBDA X; IF NOT(LENGTH(X)=Y) THEN FALSE->C
       CLOSE; END);
       IF C=FALSE THEN GOTO LOOP CLOSE;
IF Y<2 THEN CRUNCH(['TOO FEW ROWS']) CLOSE;
       APPLIST(CHITABLE, LAMBDA X; APPLIST(X, LAMBDA N; IF NOT
         (N.ISINTEGER ) OR  NOT(N>=0)  THEN FALSE->C; CLOSE; END) END);
       IF C=FALSE THEN CRUNCH(['REQUIRE POSITIVE INTEGERS']) EXIT
CHISQ(CHITABLE)->CHILIST; NL(2);
IF CHILIST.HD.ISNUMBER THEN
PR('CHI SQUARE =`::(CHILIST.HD::('  DF =`::CHILIST.TL)));NL(1);
ELSE PR(CHILIST);CLOSE; NL(2);
END


FUNCTION INSERT X NEWLIST LIST;
  NEWLIST<->(LIST.HD::(X::LIST.TL))
END


FUNCTION REPLACE X NEWLIST LIST;
  NEWLIST<->(X::LIST.TL)
END


FUNCTION EDITLIST LIST NTH X EDITFUN;
  VARS NEWLIST ENDNEWL NEXTVAL NTH;
IF LIST.NULL THEN X::NIL EXIT
  NIL::NIL -> NEWLIST;
  NEWLIST -> ENDNEWL;
  LOOP: IF LIST.NULL THEN NEWLIST.TL EXIT;
  LIST.HD -> NEXTVAL;
  IF NTH = 1 THEN IF  NOT(X=NIL) THEN EDITFUN(X,NEWLIST.TL,LIST) EXIT;
  NEWLIST.TL<->LIST.TL EXIT
  NEXTVAL::NIL -> ENDNEWL.TL;
  ENDNEWL.TL -> ENDNEWL;
  LIST.TL -> LIST;
  NTH - 1 -> NTH;
  GOTO LOOP
END


FUNCTION GOBBLE;
 L1:
  IF .R1=TERMIN THEN EXIT
  GOTO L1
END;


FUNCTION READTAPE FILENAME;
VARS LIST OBJECT LASTOB;
    NIL->LIST; "END"->LASTOB;
   POPMESS([PTIN 20]::[%FILENAME%]).INCHARITEM -> R1;
LOOP:
   R1()->OBJECT;
IF OBJECT=TERMIN THEN
   IF NOT(LASTOB="END") THEN REP(['ERROR IN TAPE']); EXIT;
EXIT
   IF OBJECT="END" THEN MATRIX<->[%LIST%]->MATRIX;NIL->LIST;
  "END"->LASTOB; GOTO LOOP CLOSE;
IF LASTOB="END"
  THEN IF OBJECT.ELIGNEW THEN GOTO LOOP1
  ELSE GOBBLE(); REP(['ERROR IN TAPE']) EXIT; CLOSE;
  IF OBJECT="-" OR OBJECT="+" THEN OBJECT->LASTOB;GOTO LOOP CLOSE;
  IF NOT(OBJECT.ISNUMBER) THEN GOBBLE();  REP(['ERROR IN TAPE']) CLOSE;
  IF LASTOB ="-" THEN -OBJECT->OBJECT; CLOSE;
LOOP1:LIST<->[%OBJECT%]->LIST;OBJECT->LASTOB; GOTO LOOP
END
```

```
FUNCTION READTAPF;
VARS REP  ERRFUN;
  JUMPOUT(LAMBDA X; X=>;NL(2) END,0)->REP;
  FUNCTION ERRFUN X N;
   IF N=55 THEN EXIT
   IF (N=59) OR (N=54) OR ( N=56)
      THEN REP(['TAPE READER IN USE:  TRY AGAIN LATER`])
      ELSEIF N=57 THEN REP(['CHECK FILENAME:`]<>X.TL.HD)
  ELSE GORBLE();  REP(['ERROR IN TAPE`]) CLOSE;
  END
 NL(2);  PR(['TYPE FILE NAME USING SINGLE WORD`]);
  READTAPE(.ITEMREAD);
END
```

```
FUNCTION NEWCOLSF;
  VARS HOLD; UNDEF->HOLD;
  LOOP:
  INCOL(IF HOLD=UNDEF THEN 'COLUMN NAME,`::SIGNOFF ELSE ['COLUMN NAME`] CLOSE );
      -> HOLD;
  IF NOT(HOLD) THEN EXIT;
  MATRIX<->[%HOLD%] -> MATRIX
  GOTO LOOP
END
```

```
FUNCTION CLEARF;
  NIL -> MATRIX;
  NIL -> TABLE
;NL(1);PR(['MATRIX NOW EMPTY`]);NL(1);
END
```

```
FUNCTION NEWNAMEF;
  VARS NEW OLD;
  INPUTCHAT(['THE NEW NAME`],NIL,ELIGNEW,['NAME USED ALREADY`],"THING") -> NEW;
  NL(1);
  INPUTCHAT(['THE OLD NAME`],NIL;ISNAME,['NO SUCH NAME`],"THING") -> OLD;
  REASSOC(NEW,OLD,MATRIX);
END
```

```
VARS EDMESS EDINPC;  NL(1);  PR('STILL COMPILING`);

['ONE OF THE WORDS: ADD, INSERT, REPLACE, DELETE`] -> EDMESS;

INPUTCHAT(%EDMESS,[ADD INSERT REPLACE DELETE],DUMMY,'WANT`::EDMESS,"THING" %)
  -> EDINPC;
```

```
FUNCTION EDITCOLF;
VARS F NTH DATAVAL COL FUN NEWCOL;
  FETCHCOL(['NAME OR NUMBER OF COLUMN TO BE EDITED`])->COL;
  EDINPC()->F;
  IF F="ADD" THEN
    COL<->INPUTCHAT('NEW VALUES,`::SIGNOFF, [END],
                   ISNUMBER, ['NOT A NUMBER`], "LIST") -> NEWCOL;
    GOTO PROUT;
  CLOSE;
  IF F="DELETE" THEN
    NIL
  ELSE
    INPUTCHAT(['NEW VALUE`], NIL, ISNUMBER, ['NOT A NUMBER`], "THING" )
  CLOSE -> DATAVAL;
```

```
IF F="INSERT" THEN INSERT->FUN ELSE REPLACE->FUN CLOSE;
INPUTCHAT('NUMBER OF ROW`::
      IF F="INSERT" THEN
       ['AFTER WHICH THE INSERTION IS TO BE MADE`]
      ELSEIF F="DELETE" THEN
       ['TO BE DELETED`]
      ELSE
       ['TO BE REPLACED`]
      CLOSE, NIL,
      LAMBDA X;
       IF X.ISINTEGER AND X<LENGTH(COL) THEN
         IF FUN=REPLACE THEN IF X>0 THEN TRUE EXIT
         ELSEIF X>=0 THEN TRUE EXIT
         CLOSE;
         FALSE
        END, ['NO SUCH ROW`], "THING") -> NTH;
  EDITLIST(COL,NTH+1,DATAVAL,FUN) -> NEWCOL;
  NUM(COL.HD,MATRIX) -> NTH;
  EDITLIST(MATRIX,NTH,NEWCOL,REPLACE)->MATRIX;
 PROUT:
  1.NL;
  PR('EDITED VERSION FOR CHECKING:

`::NEWCOL);
  NL(2);
END;
```

```
FUNCTION EDITTABF;
VARS F OLDCOL DATACOL FUN;
  EDINPC() -> F;
  IF F = "DELETE" THEN NIL -> DATACOL;
  ELSE INCOL(['NEW COLUMN NAME`]) -> DATACOL CLOSE;
  IF F = "ADD" OR F= "INSERT" THEN INSERT->FUN
  ELSE REPLACE ->FUN
CLOSE;
  IF F="ADD" THEN LENGTH(MATRIX) ->OLDCOL;
  ELSE INPUTCHAT('NUMBER OF COLUMN`::IF F = "INSERT" THEN
  ['AFTER WHICH INSERTION TO BE MADE`] ELSEIF
  F = "DELETE" THEN ['TO BE DELETED`]
  ELSEIF F="REPLACE" THEN  ['TO BE REPLACED`]
CLOSE;,
  NIL,INBOUNDS,['NO SUCH COLUMN`],"THING") -> OLDCOL CLOSE;
  EDITLIST(MATRIX,OLDCOL,DATACOL,FUN) -> MATRIX;
  IF F="DELETE" THEN EXIT;
END
```

```
FUNCTION SUBTABF;
  VARS COLUMNS;
  NIL -> COLUMNS;
  NIL -> TABLE;
IF MATRIX.NULL THEN NL(1);PR(['NO DATA IN MATRIX`]);NL(1);EXIT;
  INPUTCHAT('COLUMN NAMES OR NUMBERS:`::SIGNOFF,[END],
  ISCOL,['NO SUCH COLUMN`],"LIST") -> COLUMNS;
  LOOP: IF COLUMNS.NULL THEN IF PRTABLE THEN COLPRINT(TABLE);CLOSE;  EXIT;
TABLE<->[%ASSOCGET(MATRIX,COLUMNS.HD)%] -> TABLE;
  COLUMNS.TL -> COLUMNS;
  GOTO LOOP
END
```

```
FUNCTION COLPRINF;
  COLPRINT(MATRIX)
END
```

```
FUNCTION COMMAND VOCAB VOCABNAME;
VARS WORD ACT WORDS ACTS HELPS QUIT CRUNCH;
  VOCAB.HD->WORDS;
  VOCAB.TL.HD->ACTS;
  VOCAB.TL.TL.HD->HELPS;
  VOCAB.TL.TL.TL.HD->QUIT;
  JUMPOUT(LAMBDA X;X=> ;NL(2) END,0)->CRUNCH;
  LOOP:INPUTCHAT(VOCABNAME
      <>['COMMAND'],WORDS,DUMMY,'TYPE ONE OF THE WORDS:'::WORDS,
      "THING")->WORD;
  IF WORD="HELP" THEN NL(1);
  ASSOCPR(HELPS); NL(1);  GOTO LOOP CLOSE;
  IF WORD=QUIT THEN NL(2) EXIT
  APPLY(ASSOCVAL(WORD,ACTS));
  GOTO LOOP
END



FUNCTION NEWCOMMAND WORD ACT HELP VOCAB;
  WORD::VOCAB.HD->VOCAB.HD;
  ASSOC(WORD,ACT,VOCAB.TL.HD)->VOCAB.TL.HD;
  ASSOC(WORD,HELP,VOCAB.TL.TL.HD)->VOCAB.TL.TL.HD
END



FUNCTION NEWVOCAB QUIT;
  [%[%"HELP",QUIT%],NIL,[%QUIT::['TO EXIT FROM THIS VOCABULARY']%],QUIT%]
END


VARS OERRF MAINVOC DATAVOC STATVOC  EDITVOC;  ERRFUN -> OERRF;



FUNCTION POPSTATS;
VARS CUCHAROUT  ERRFUN;
  FUNCTION ERRFUN X N;
    IF N=33 THEN CRUNCH(['CALCULATION INVOLVES A NUMBER TOO LARGE TO HANDLE']); NL(1);
ELSEIF N=37 THEN CRUNCH(['PROCEDURE ASKED FOR IS NOT DEFINED']);NL(1);
    ELSEIF N=39 OR N=40 OR N=41 OR N=42 OR N=43 OR N=44 OR N=47
THEN CRUNCH(['CALCULATION INVOLVES ITEM UNSUITABLE FOR THE DESIRED ARITHMETIC OPERATI(
ELSE   OERRF(X,N) CLOSE;
  END;
  0 -> CHARCOUNT;  FORM -> CUCHAROUT;
  NL(2);
  COMMAND(MAINVOC,[POPSTATS])
END




FUNCTION DATACOMF;
VARS CRUNCH;JUMPOUT(LAMBDA X;X=>;NL(2);END,0)->CRUNCH; TRUE->PRTABLE;
  COMMAND(DATAVOC,[DATA])
END



FUNCTION STATCOMF;
VARS CRUNCH;
JUMPOUT(LAMBDA X;X=>;NL(2);END,0)->CRUNCH;
IF MATRIX.NULL THEN NL(1);PR(['NO DATA IN MATRIX']);NL(1);EXIT;

  COMMAND(STATVOC,[STATS])
END



FUNCTION EDITCOMF;
VARS CRUNCH;
JUMPOUT(LAMBDA X;X=>;NL(2);END,0)->CRUNCH;
IF MATRIX.NULL THEN NL(2);PR(['NO DATA IN MATRIX']);NL(2);EXIT
  COMMAND(EDITVOC,[EDIT])
END
```

```
FUNCTION DROPCOMF;
VARS NAME VOCABNAME COUNT Y CRUNCH;
  0->COUNT; NIL->Y;
JUMPOUT(LAMBDA X;X=>;NL(2); END,0)->CRUNCH;
INPUTCHAT(['NAME FOR COMMAND TO BE DROPPED`],MAINVOC.HD<>
  DATAVOC.HD<>STATVOC.HD<>EDITVOC.HD, DUMMY,['NO SUCH COMMAND`],
  "THING")->NAME;
LOOP: INPUTCHAT(['NAME OF VOCABULARY-MAIN,DATA,STAT OR EDIT`],
     [MAIN DATA STAT EDIT],DUMMY,['NOT A VOCABULARY NAME`],
     "THING")->VOCABNAME;
  IF VOCABNAME="MAIN" THEN MAINVOC
  ELSEIF VOCABNAME ="DATA" THEN DATAVOC
  ELSEIF VOCABNAME="EDIT" THEN EDITVOC
  ELSE STATVOC CLOSE; -> VOCABNAME;
IF NOT(MEMBER(NAME,VOCABNAME.HD)) THEN COUNT+1->COUNT;
IF COUNT>2 THEN CRUNCH(['WRONG VOCABULARY NAME`])
ELSE PR(['WRONG VOCABULARY NAME`]) GOTO LOOP CLOSE; CLOSE;
  DROP(NAME, VOCABNAME.HD)->VOCABNAME.HD;
IF VOCABNAME="MAINVOC" THEN MAINVOC.HD->WORDS;CLOSE;
  VOCABNAME.TL.TL.HD->Y;
DROP(ASSOCGET(Y,NAME),Y)->L;
IF VOCABNAME ="MAINVOC" THEN Y->HELPS; CLOSE;
END

FUNCTION NEWCOMF;
VARS WORD NAME FUN HELPMESS VOCABNAME CRUNCH;
      JUMPOUT(LAMBDA X; X=> NL(2) END,0) -> CRUNCH;
     INPUTCHAT(['NAME FOR NEW COMMAND`],NIL,LAMBDA X;IF NOT(X.ISNUMBER)
     AND NOT(MEMBER(X,[POPSTATS]<>MAINVOC.HD<>DATAVOC.HD<>STATVOC.HD<>
     EDITVOC.HD)) THEN TRUE ELSE FALSE CLOSE; END,['NOT SUITABLE`],
     "THING")->NAME;
INPUTCHAT(['NAME OF FUNCTION TO BE CALLED`],NIL,LAMBDA X;
   ISFUNC(POPVAL(X::[;GOON])) END,['NO SUCH FUNCTION`],"THING")->WORD;
   POPVAL(WORD::[;GOON])->FUN;
INPUTCHAT(['MESSAGE TO EXPLAIN ACTION OF NEW COMMAND:`]<>SIGNOFF,NIL,GOOD,
NIL,"LIST") ->HELPMESS;
INPUTCHAT(['WHICH VOCAB - MAIN, DATA, STAT OR EDIT FOR NEWCOMMAND`],
     [MAIN DATA STAT EDIT],DUMMY,['NOT A VOCABNAME`],"THING")->VOCABNAME;
NEWCOMMAND(NAME,FUN,HELPMESS,IF VOCABNAME ="MAIN" THEN MAINVOC
     ELSEIF VOCABNAME="DATA" THEN DATAVOC ELSEIF VOCABNAME="STAT"
     THEN STATVOC ELSE EDITVOC CLOSE;);
IF VOCABNAME="MAIN" THEN MAINVOC.HD->WORDS; CLOSE;
END

NEWVOCAB("POP2")->MAINVOC;
NEWVOCAB("END")->DATAVOC;
NEWVOCAB("END")->STATVOC;
NEWVOCAB("END")->EDITVOC;

NEWCOMMAND("EDIT",EDITCOMF,['TO EDIT THE DATA`],MAINVOC);
NEWCOMMAND("DATA",DATACOMF,['TO INPUT AND OUTPUT DATA`],MAINVOC);
NEWCOMMAND("STATS",STATCOMF,['TO COMPUTE STATISTICS OF DATA`],MAINVOC);
NEWCOMMAND("NEWCOM",NEWCOMF,['TO ADD A NEW COMMAND`],MAINVOC);
NEWCOMMAND("DROPCOM",DROPCOMF,['TO DROP COMMAND FROM VOCABULARY`],MAINVOC);
NEWCOMMAND("CLEAR",CLEARF,['TO CLEAR ALL DATA`],DATAVOC);
NEWCOMMAND("NEWNAME",NEWNAMEF,['TO GIVE A COLUMN A NEW NAME`],EDITVOC);
NEWCOMMAND("NEWCOLS",NEWCOLSF,['TO TYPE IN SOME DATA COLUMNS`],DATAVOC);
NEWCOMMAND("EDITTAB",EDITTABF,['TO EDIT WHOLE COLUMNS`],EDITVOC);
NEWCOMMAND("EDITCOL",EDITCOLF,['TO EDIT VALUES IN A COLUMN`],EDITVOC);
NEWCOMMAND("COLPRINT",COLPRINF,
          ['TO PRINT THE DATA COLUMN-WISE`],DATAVOC);
NEWCOMMAND("SUBTAB",SUBTABF,
          ['TO FORM AND PRINT A SUBTABLE OF THE DATA`],DATAVOC);

NEWCOMMAND("GENSTAT",GENSTATF,['TO COMPUTE GENERAL STATS`],STATVOC);
NEWCOMMAND("TTEST",TTESTF,['TO APPLY T TEST`],STATVOC);
NEWCOMMAND("CORREL",CORRELF,['CORRELATION BETWEEN TWO COLUMNS`],STATVOC);
NEWCOMMAND("RESCALE",RESCALEF,
          ['TO RESCALE COLUMN BY APPLYING TRANSFORMATION`],STATVOC);
NEWCOMMAND("HISTO",HISTOF,['TO DRAW A HISTOGRAM`],STATVOC);
NEWCOMMAND("CONTING",CONTINGF,
      ['TO PERFORM A CHI-SQUARED TEST`],STATVOC);
NEWCOMMAND("READTAPE",READTAPF,['TO INPUT DATA FROM PAPER TAPE`],DATAVOC);

PR('

POPSTATS NOW COMPILED.

TO ENTER PROGRAM TYPE  .POPSTATS;
`);  NL(2);
```

Program name. LIB PROOF CHECKER
Source. D.B.Anderson,DMIP; *Date of issue.* March 1970

¶*Description.* Since 1965 Robinson's resolution method of theorem proving in first order logic has been widely used. The purpose of this POP-2 program is to do binary resolution and factoring of clauses in this formulation, and simple facilities have been added so that it can be used as a proof-checker.

¶*How to use the program.* (1) Compile the program by typing:
COMPILE(LIBRARY([LIB PROOF CHECKER]));

(2) Declare all variable, constant, function, and predicate names. These are all POP-2 words so that only the first eight characters are significant.
⟨arity⟩ ::= non-negative integer
⟨namelist⟩ ::= list of POP-2 words

VARBS ⟨namelist⟩;	e.g. VARBS [X Y Z];
CONSTS ⟨namelist⟩;	e.g. CONSTS [A BILL];
⟨arity⟩ FUNS ⟨namelist⟩;	e.g. 1 FUNS [F];

⟨arity⟩ PREDS ⟨namelist⟩;	e.g. 1 PREDS [P];
	2 PREDS [Q];
	3 PREDS [R];

Names may be declared at any time before they are used in a clause.

(3) Put in some clauses, using the function ADDCLAUSE. Clauses are POP-2 list expressions, the elements of the list being literals using the standard terminology and notation. —— is used for the *not* prefix. For example,
ADDCLAUSE([%——R(Y, Z, A), Q(X, Y), Q(X, Z)%]);
The program responds with a message:
[OK NUMBER 1],
the number given being the index of the clause, which has just been added to CLS, the array of clauses, that is,
CLS(1) is now the clause [——R(Y, Z, A) Q(X, Y) Q(X, Z)].
ADDCLAUSE([%Q(X, F(A))%]);
[OK NUMBER 2]
ADDCLUASE([%——Q(Z, Y), P(Y)%]);
[OK NUMBER 3]
ADDCLAUSE([%——P(X)%]);
[OK NUMBER 4]
There is no syntax check in the program so that if the POP-2 compiler is satisfied there will be no error message at this stage. Be careful to give functions the right number of arguments.

(4) Any clause may be printed out using the function PRR, for example,
PRR(CLS(1));
[——R(Y, Z, A) Q(X, Y) Q(X, Z)]:

(5) Two clauses can be resolved on given literals using function RESOLVE.

RESOLVE ∊ clause index, literal number, clause index, literal number => ( ) For example,
RESOLVE(3, 1, 4, 2) resolves the 3rd and 4th clauses upon the 1st literal of 3 and the 2nd literal of 4.
: RESOLVE(2, 1, 3, 1);
[P(F(A))] [OK NUMBER 5]

If the resolution is possible, the resolvent is printed out and ADDCLAUSEd to the array CLS. If the literals do not unify, a message is given.

```
: RESOLVE(2, 1, 1, 2);
[SORRY – WON'T RESOLVE]
```

(6) A clause can be factored with respect to a given pair of literals. FACTOR ϵ clause index, literal number, literal number => (). For example,

```
FACTOR(1, 2, 3);
[−−R(Z, Z, A) Q(X, Z)] [OK NUMBER 6]
FACTOR(1, 1, 3);
[SORRY – WON'T FACTOR]
```

(7) The empty clause is called NIL.

```
RESOLVE(4, 1, 5, 1);
NIL [OK NUMBER 7]
```

(8) The instantiation dictionary used in the latest unification (either in resolution or factoring) is called DICT.

```
PRR(DICT);
[[X. F(A)]]:
```

(9) The POP-2 functions used in the program provide a basis for the writing of other functions enabling the user to overwrite clauses, increase the number of clauses, record ancestors of clauses, print out the proof trees, and so on.

Warning. There are very few diagnostic error messages in this skeletal system— do not ask for the tenth literal of a clause of length four and do not ask for the millionth clause. Check your syntax carefully. Note that the arguments of RESOLVE and FACTOR are in a different order from that in earlier versions of the program.

¶Method used. A function map is given in figure 1 and the program listing contains comments.

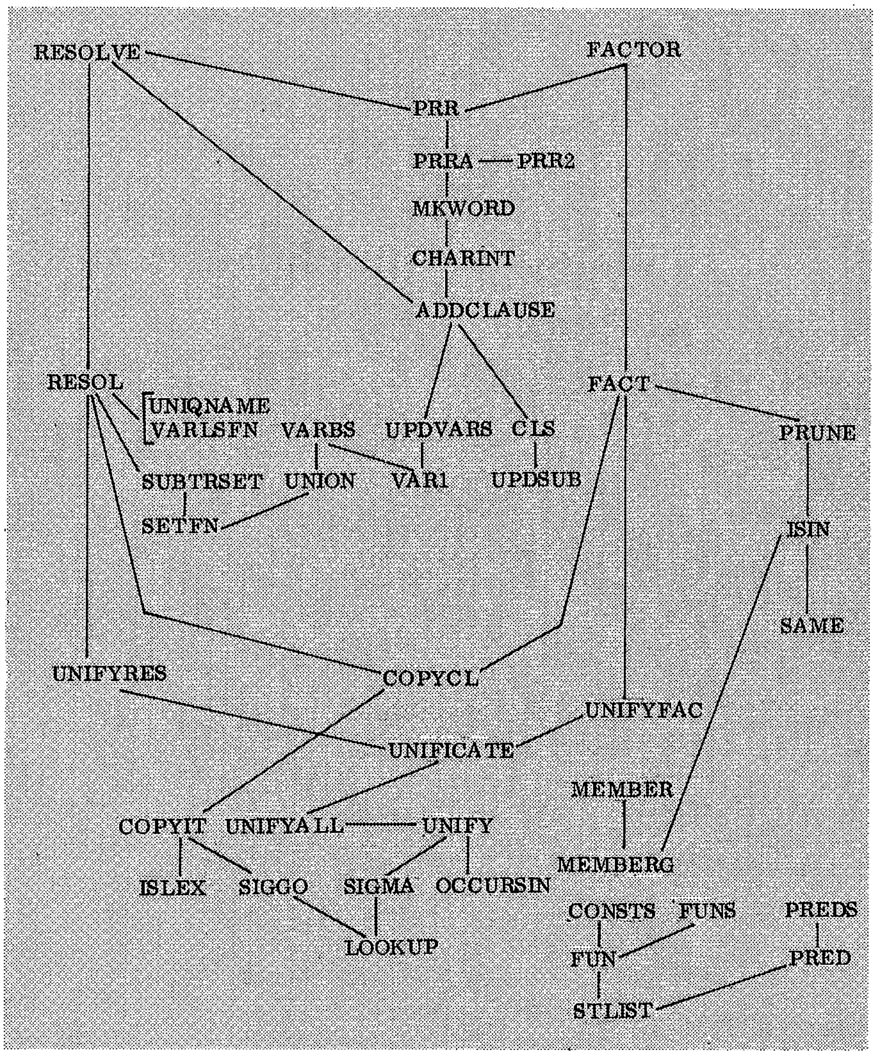
The clauses are represented in POP-2 by record structures mirroring closely their syntactic structure.

```
A clause is a list of PREDS
A PRED has a SIGN              (++or −−)
         and a NAMEP           (POP-2 word)
         and an ARGLISTP       (list of FUNS or VARS)
A FUN has a NAMEF
        and an ARGLISTF
A VARS has a NAMEV
         and a TAG             (integer)
A constant is a function of no arguments.
```

See figure 2 for an example. Note that in any clause there is only one copy of any variable, correctly reflecting the clause structure.

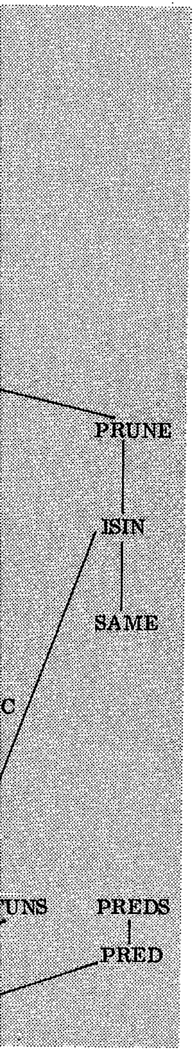
When the program is compiled, a full strip of 30 items CLSTRIP is made, with indexing function CLS.

Operator VARBS adds the new variable names to the current list VARLIST, rejecting duplicates, and constructs the corresponding data structures. The function UPDVARS causes a new set of structures to be made for the variable (the TAG value is equal to the index number in CLS) and this is done in every ADDCLAUSE so that the variables in two clauses are never the same.



RECORDFNS and uses of TESTER
and MEMBER have been omitted.

FIGURE 1


PRUNE
ISIN
SAME
C
UNS
PREDS
PRED

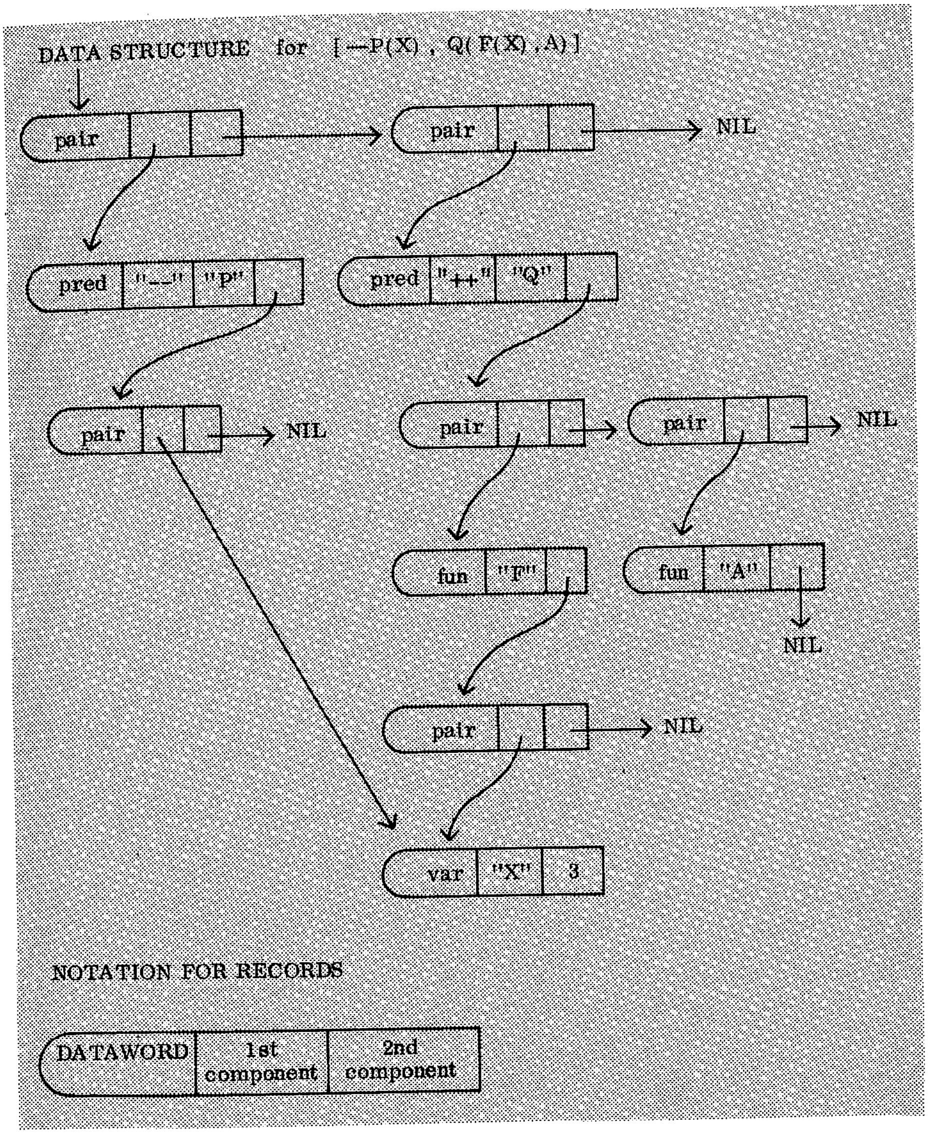
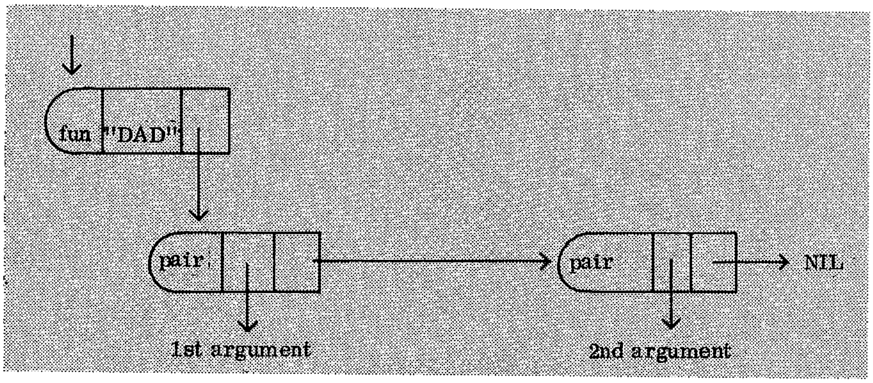

DATA STRUCTURE for [—P(X) , Q(F(X) , A)]
pair
pair
NIL
pred
"--"
"P"
pred
"++"
"Q"
pair
NIL
pair
pair
NIL
fun
"F"
fun
"A"
NIL
pair
NIL
var
"X"
3
NOTATION FOR RECORDS
DATAWORD
1st component
2nd component

FIGURE 2

Operator FUNS uses function FUN to make the words in its list argument into the identifiers of the corresponding record constructor functions: for example,
: 2 FUNS DAD ;
means that DAD is now a function of two arguments which constructs a record as shown below when it is called. Function STLIST is used to put the required number of arguments in the list.



FUNS with arity 0 is CONSTS, and in this case the identifiers are *operators* so that brackets are not needed to cause execution.

PREDS is like FUNS. The SIGNS are made positive (++) and operator —— makes them negative (——) if it is used.

ADDCLAUSE puts its argument (a clause) into the strip CLSTRIP, outputs the index message, updates its frozen-in pointer to the next empty space in CLSTRIP, and calls UPDVARS.

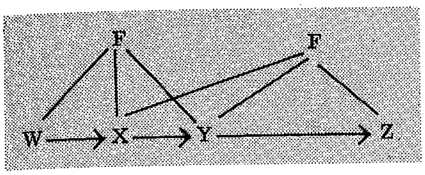
PRR prints out a clause in a format similar to the input format. Empty pairs of brackets after constants are avoided. Calling function SEE alternately causes TAGS of variables to be printed or not to be printed with their names. This is useful in debugging.

Resolution and factoring are basically rather similar as they both involve unification of two literals. The unification algorithm is that given by Burstall (1969), very similar to the original one of Robinson (1965), and is highly recursive. Functions UNIFY and UNIFYALL unify terms—UNIFICATE unifies literals and, depending on the truth value of its third argument does resolution (UNIFYRES) or factoring (UNIFYFAC). During the unification attempt all substitutions are put in a dictionary DICT whose lookup function SIGMA is continually used to update the structures. It is for this reason that copies of clauses are taken in FACT and RESOL (using COPYCL) and it is these that are operated upon.

DICT can be printed out with PRR and shows the substitutions (instantiations) made in the last unification.

In resolution the problem may arise that two variables in the resolvent have the same name though of course their TAGS are different as they must be different variables. Function VARLSFN puts the variables of a clause into VARSLIST and UNIQNAME renames any clashing ones from the spare variable list SPVLIST which is VARLIST-VARSLIST. These variables have their TAGS set to 0 so that the action of UNIQNAME is seen if the TAGS are displayed.

In factoring the dictionary function SIGMA must be used repeatedly after unification. This is because the dictionary DICT may contain strings of substitutions which must be applied one after another to a variable. For example,
{F(w, x, y), F(x, y, z)}



DICT is [y::z x::y w::x] and the factor F(z, z, z) is reached only after 3 uses of SIGMA.

After 0	F(w, x, y)	F(x, y, z)
1	F(x, y, z)	F(y, z, z)
2	F(y, z, z)	F(z, z, z)
3	F(z, z, z)	F(z, z, z)

Function PRUNE then removes all duplicates: it uses SAME, the equality function for literals.

RESOLVE and FACTOR do the resolution and factoring (using RESOL and FACT) and also output messages and ADDCLAUSE the resolvents or factors.

¶*Global variables. Type Operation.* FUNS PREDS CONST —— VARBS

Type General. TAG NAMEV DESTVAR CONSVAR ARGLISTP NAMEP SIGN DESTPRED CONSPRED ARGLISTF NAMEF DESTFUN CONSFUN ISFUN ISVAR ISPRED ISPAIR VARLIST VARSLIST SPVLIST DICT DICTO CLSTRIP CLS UPDSUB MEMBER ISIN UNIFYRES UNIFYFAC SUBTRSET UNION ADDCLAUSE PRR2 PRR UNIFYALL UNIFICATE UNIFYRES UNIFYFAC SAME ISIN PRUNE UNIQNAME VARLSFN GET REMOVE NCJ RESOL RESOLVE TESTER STLIST PRED FUN SETFN SUBTRSET UNION VAR1 UPDVARS CHARINT MKWORD MEMBERG MEMBER FACT ISNEG PRRA PRR2 PRR SEE LOOKUP SIGMA SIGGO ISLEX COPYIT COPYCL OCCURSIN UNIFY FACTOR

¶*Store used.* The program uses 11 blocks of store.

¶*Example of use.*

```
: [PROOF CHECKER]. LIBRARY. COMPILE;
:
:
:  VARBS [S S1 S2 S3];
:  CONSTS [A B C D S0];
:  3 FUNS [MOVE];
:  1 PREDS [ANSWER];
:  2 PREDS [AT];
:
:  ADDCLAUSE([% ——AT(C, S), ANSWER(S) %]);
[OK NUMBER 1]
:  ADDCLAUSE([% ——AT(B, S3), AT(C, MOVE(B, C, S3)) %]);
[OK NUMBER 2]
:  RESOLVE(1, 1, 2, 2);
[ANSWER(MOVE(B, C, S3)) ——AT(B, S3)][OK NUMBER 3]
:  ADDCLAUSE([% ——AT(A, S1), AT(B, MOVE(A, B, S1)) %]);
[OK NUMBER 4]
:  RESOLVE(2, 3, 2, 4);
```

```
[ANSWER(MOVE(B, C, MOVE(A, B, S1))) ——AT(A, S1)][OK NUMBER 5]
: ADDCLAUSE([% AT(A, S0) %]);
[OK NUMBER 6]
: RESOLVE(2, 5, 1, 6);
[ANSWER(MOVE(B, C, MOVE(A, B, S0)))][OK NUMBER 7]
```

REFERENCES

Burstall, R. M. (1965) Computer theorem proving. *Research Memorandum EPU-R-5.* Department of Machine Intelligence and Perception, University of Edinburgh.

Burstall, R. M. (1969) Notes to Diploma class, Department of Machine Intelligence and Perception, University of Edinburgh.

Green, C. (1969) Theorem-proving by resolution as a basis for a question-answering system. *Machine Intelligence 4,* pp. 183-205 (eds. B. Meltzer & D. Michie). Edinburgh: Edinburgh University Press.

Luckham, D.. (1967) The resolution principle in theorem-proving. *Machine Intelligence 1,* pp. 47-61 (eds. N. L. Collins & D. Michie). Edinburgh: Edinburgh University Press.

Luckham, D. (1968) Some tree-paring strategies for theorem-proving. *Machine Intelligence 3,* pp. 95-112 (ed. D. Michie). Edinburgh: Edinburgh University Press.

Robinson, J. A. (1965) A machine-oriented logic based on the resolution principle. *J. Ass. comput Mach., 12,* 23-41

```
[PROOF CHECKER]

VARS TAG NAMEV DESTVAR CONSVAR ARGLISTP NAMEP SIGN DESTPRED
CONSPRED ARGLISTF NAMEF DESTFUN CONSFUN ISFUN ISVAR ISPRED ISPAIR
VARLIST VARSLIST SPVLIST DICT DICTO CLSTRIP CLS UPDSUB MEMBER
ISIN UNIFYRES UNIFYFAC SUBTRSET UNION UNIFYALL ADDCLAUSE PRR2 PRR;



NIL->VARLIST;

RECORDFNS("VAR",[0 0])->TAG ->NAMEV ->DESTVAR ->CONSVAR;
RECORDFNS("PRED",[0 0 0])->ARGLISTP ->NAMEP ->SIGN
                                ->DESTPRED ->CONSPRED;
RECORDFNS("FUN",[0 0])->ARGLISTF ->NAMEF ->DESTFUN ->CONSFUN;

FUNCTION TESTER NODE WORD; DATAWORD(NODE)=WORD; END
TESTER(%"FUN"%)->ISFUN;
TESTER(%"VAR"%)->ISVAR;
TESTER(%"PRED"%)->ISPRED;
TESTER(%"PAIR"%)->ISPAIR;


FUNCTION STLIST N;
COMMENT 'PRODUCES A LIST OF THE TOP N ITEMS OF THE STACK';
NIL;
LOOP: IF N>0 THEN N-1->N; :: ; GOTO LOOP ELSE EXIT;
END


FUNCTION PRED NAME ARITY;
LAMBDA NUM NOM;
VARS LIST; STLIST(NUM)->LIST;
CONSPRED("++",NOM,LIST);
END(%ARITY,NAME%);
IF ARITY>0 THEN ->VALOF(NAME)
ELSE POPVAL([% "CANCEL",NAME,";","VARS","OPERATION",1,NAME,";",
"->","NONOP",NAME,";",
           "GOON" %]);
CLOSE;
END


FUNCTION FUN NAME ARITY;
LAMBDA NUM NOM;
VARS LIST; STLIST(NUM)->LIST;
CONSFUN(NOM,LIST);
 END(%ARITY,NAME%);
IF  ARITY>0 THEN ->VALOF(NAME)
ELSE POPVAL([% "CANCEL",NAME,";", "VARS","OPERATION",1,NAME,";",
"->","NONOP",NAME,";",
       "GOON" %]);
CLOSE;
END

VARS OPERATION 3 (FUNS PREDS CONSTS);


LAMBDA ARITY LIST;
APPLIST(LIST,FUN(%ARITY%))
END->NONOP FUNS;


LAMBDA ARITY LIST;
APPLIST(LIST,PRED(%ARITY%))
END->NONOP PREDS;


LAMBDA LIST;
APPLIST(LIST,FUN(%0%))
END->NONOP CONSTS;


VARS OPERATION 7 --;
LAMBDA PPP;
IF PPP.ISPRED THEN "--"->SIGN(PPP); PPP
             ELSE 'X--'.PR; SETPOP()
CLOSE;
END->NONOP --;
```

```
FUNCTION SETFN A303 B303 T1 T2;
COMMENT 'GENERAL SET THEORY FUNCTION';
VARS X303 C303;
IF T1 THEN B303->C303; ELSE NIL->C303 CLOSE;
LOOP: IF A303.NULL THEN C303
      ELSE A303.NEXT->A303->X303;
         IF MEMBER(X303,B303)=T2 THEN X303::C303->C303 CLOSE;
         GOTO LOOP
     CLOSE;
END

 SETFN(%FALSE,FALSE%)->SUBTRSET;
 SETFN(%TRUE,FALSE%)->UNION;


FUNCTION VAR1 VARNAME INDEX;
COMMENT 'CONSTRUCTS VARS RECORDS';
CONSVAR(VARNAME,INDEX)->VALOF(VARNAME);
END


VARS OPERATION 3 VARBS;

LAMBDA LIST;
UNION(LIST,VARLIST)->VARLIST;
APPLIST(VARLIST,VAR1(%FROZVAL(1,ADDCLAUSE)%));
END->NONOP VARBS;

FUNCTION UPDVARS N;
COMMENT 'MAKES NEW SET OF VARIABLES';
APPLIST(VARLIST,VAR1(%N%));
END


FUNCTION CHARINT INT;
COMMENT 'MAKES AN INTEGER INTO CHARACTERS ON THE STACK';
VARS N LIS; NIL->LIS; 0->N;
LOOP: ((INT//10)->INT)::LIS->LIS;
       N+1->N;
      IF INT>0 THEN GOTO LOOP
           ELSE APPLIST(LIS,LAMBDA X; X END); N
      CLOSE;
END


FUNCTION MKWORD WORD INTEGER;
COMMENT 'MAKES A WORD FROM A WORD AND AN INTEGER';
VARS L1 L2;
DESTWORD(WORD)->L1; CHARINT(INTEGER)->L2;
IF L1+L2>8 THEN 'XMKWORD'.PR; SETPOP() ELSE L1+L2; CONSWORD()
CLOSE;
END



FUNCTION MEMBERG OBJ LOBBLE EQFN;
COMMENT 'GENERAL MEMBERSHIP PREDICATE';
IF LOBBLE.NULL THEN FALSE
ELSEIF EQFN(OBJ,LOBBLE.HD) THEN TRUE
ELSE MEMBERG(OBJ,LOBBLE.TL,EQFN)
CLOSE;
END



MEMBERG(%NONOP = %)->MEMBER;

FUNCTION ISNEG PREDCATE;
SIGN(PREDCATE)="--"
END


FUNCTION PRRA ITEM TVIND;
COMMENT 'THE PRINTING FUNCTION';
    IF ITEM.ISNUMBER THEN ITEM.PR
ELSEIF ITEM.ISVAR THEN
          IF TVIND THEN MKWORD(NAMEV(ITEM),TAG(ITEM)).PR
                   ELSE NAMEV(ITEM).PR
          CLOSE;
```

```
ELSEIF ITEM.ISFUN THEN NAMEF(ITEM).PR;
          IF ARGLISTF(ITEM).NULL.NOT THEN
           "(".PR; PRR2(ARGLISTF(ITEM),TVIND); ")".PR;
          CLOSE;
ELSEIF ITEM.ISPRED THEN
          IF ITEM.ISNEG THEN "--".PR CLOSE;
          NAMEP(ITEM).PR;
          IF ARGLISTP(ITEM).NULL.NOT THEN
           "(".PR; PRR2(ARGLISTP(ITEM),TVIND); ")".PR; CLOSE;
ELSEIF ITEM.ISPAIR THEN "[".PR;
          IF ITEM.BACK.ISPAIR OR ITEM.BACK="NIL"
            THEN APPLIST(ITEM,LAMBDA X; PRRA(X,TVIND); SP(1) END)
           ELSE PRRA(ITEM.FRONT,TVIND); ".".PR;
                    PRRA(ITEM.BACK,TVIND);          CLOSE;
                        "]".PR
ELSE ITEM.PR
CLOSE;
END

FUNCTION PRR2 LIST TVIND;
IF LIST.NULL.NOT THEN PRRA(LIST.HD,TVIND);
      APPLIST(LIST.TL,LAMBDA X; ",".PR; PRRA(X,TVIND) END);
CLOSE;
END

PRRA(%FALSE%)->PRR;

FUNCTION SEE;
COMMENT 'SWITCH FOR TAG DISPLAY';
NOT(FROZVAL(1,PRR))->FROZVAL(1,PRR);
END

FUNCTION LOOKUP A303 DICT EQFN;
COMMENT 'SIMPLE DICTIONARY LOOKUP';
IF NULL(DICT) THEN A303
ELSEIF EQFN(DICT.HD.FRONT,A303) THEN DICT.HD.BACK
ELSE LOOKUP(A303,DICT.TL)
CLOSE;
END

LOOKUP(%NONOP = %)->LOOKUP;



FUNCTION SIGMA E;
COMMENT 'DOES SUBSTITUTIONS(INSTANTIATIONS) IN UNIFICATION';
    IF E.ISVAR THEN LOOKUP(E,DICT)
ELSEIF E.ISFUN THEN MAPLIST(ARGLISTF(E),SIGMA)->ARGLISTF(E); E
ELSEIF E.ISPRED THEN MAPLIST(ARGLISTP(E),SIGMA)->ARGLISTP(E); E
ELSEIF E.ISPAIR THEN MAPLIST(E,SIGMA);
ELSEIF E=NIL THEN NIL;
CLOSE;
END


FUNCTION SIGGO X;
LOOKUP(X,DICTO)
END

FUNCTION ISLEX X;
COMMENT 'HAS THIS X ALREADY BEEN COPIED';
MEMBER(X,MAPLIST(DICTO,FRONT))
END

FUNCTION COPYIT ITEM;
COMMENT 'COPIES LITERALS';
VARS HERE;
IF ITEM.ISVAR THEN
          IF ITEM.ISLEX THEN SIGGO(ITEM)
               ELSE CONSVAR(NAMEV(ITEM),TAG(ITEM))->HERE;
                   (ITEM::HERE)::DICTO->DICTO; HERE
         CLOSE;
ELSEIF ITEM.ISFUN THEN
          CONSFUN(NAMEF(ITEM),MAPLIST(ARGLISTF(ITEM),COPYIT))
ELSEIF ITEM.ISPRED THEN
CONSPRED(SIGN(ITEM),NAMEP(ITEM),MAPLIST(ARGLISTP(ITEM),COPYIT))
CLOSE;
END
```

```
FUNCTION COPYCL CLAUSE;
COMMENT 'COPIES CLAUSES';
NIL->DICTO;
MAPLIST(CLAUSE,COPYIT)
END


FUNCTION OCCURSIN E2 E1;
COMMENT 'DOES E1 OCCUR IN E2';
IF E2.ISVAR THEN E1=E2
ELSEIF E2.ISFUN THEN MEMBER(1,MAPLIST(ARGLISTF(E2),OCCURSIN(%E1%)))
ELSEIF E2.ISPRED THEN MEMBER(1,MAPLIST(ARGLISTP(E2),OCCURSIN(%E1%)))
CLOSE;
END


FUNCTION UNIFY E1 E2;
COMMENT 'UNIFIES TERMS';
SIGMA(E1)->E1; SIGMA(E2)->E2;
IF E1.ISVAR AND E2.ISVAR THEN
     IF E1=E2 THEN TRUE
              ELSE TRUE; (E1::E2)::DICT->DICT;
     CLOSE;
ELSEIF E1.ISVAR AND E2.ISFUN THEN
     IF OCCURSIN(E2,E1) THEN FALSE
                        ELSE TRUE; (E1::E2)::DICT->DICT;
     CLOSE;
ELSEIF E1.ISFUN AND E2.ISVAR THEN
     IF OCCURSIN(E1,E2) THEN FALSE
                        ELSE TRUE; (E2::E1)::DICT->DICT;;
     CLOSE;
ELSEIF E1.ISFUN AND E2.ISFUN THEN
     IF NOT(NAMEF(E1)=NAMEF(E2)) THEN FALSE
               ELSE UNIFYALL(ARGLISTF(E1),ARGLISTF(E2))
     CLOSE; ELSE FALSE

CLOSE;
END


FUNCTION UNIFYALL EL1 EL2;
COMMENT 'UNIFIES TWO LISTS OF TERMS';
IF NULL(EL1) AND NULL(EL2) THEN TRUE
ELSEIF UNIFY(EL1.HD,EL2.HD) THEN UNIFYALL(EL1.TL,EL2.TL)
ELSE FALSE
CLOSE;
END


FUNCTION UNIFICATE L1 L2 TVAL;
COMMENT 'UNIFIES TWO LITERALS';
NIL->DICT;
IF L1.ISPRED AND L2.ISPRED AND NAMEP(L1)=NAMEP(L2)
AND (SIGN(L1)=SIGN(L2))=TVAL THEN
 UNIFYALL(ARGLISTP(L1),ARGLISTP(L2))
  ELSE FALSE CLOSE;
END

UNIFICATE(%FALSE%)->UNIFYRES;
UNIFICATE(%TRUE%)->UNIFYFAC;


FUNCTION SAME L1 L2;
COMMENT 'EQUALITY PREDICATE FOR LITERALS';
IF L1.ISPRED AND L2.ISPRED AND SIGN(L1)=SIGN(L2)
AND NAMEP(L1)=NAMEP(L2) THEN SAME(ARGLISTP(L1),ARGLISTP(L2));
ELSEIF L1.ISFUN AND L2.ISFUN AND NAMEF(L1)=NAMEF(L2) THEN
    SAME(ARGLISTF(L1),ARGLISTF(L2));
ELSEIF L1.ISVAR AND L2.ISVAR THEN L1=L2;
ELSEIF L1=NIL AND L2=NIL THEN TRUE;
ELSEIF L1.ISPAIR AND L2.ISPAIR AND SAME(L1.HD,L2.HD)
     THEN SAME(L1.TL,L2.TL);
ELSE FALSE
CLOSE;
END

MEMBERG(%SAME%)->ISIN;
```

```
FUNCTION PRUNE SET;
COMMENT 'PRUNES SET OF ANY DUPLICATES`;
IF SET.NULL THEN NIL
ELSEIF ISIN(SET.HD,SET.TL) THEN PRUNE(SET.TL)
ELSE SET.HD::PRUNE(SET.TL)
CLOSE;
END



FUNCTION UNIQNAME LIST;
COMMENT 'GIVES UNIQUE NAME TO EACH VARIABLE OF CLAUSE`;
VARS OLD;
IF LIST.NULL THEN RETURN
ELSEIF MEMBER(NAMEV(LIST.HD),MAPLIST(LIST.TL,NAMEV))
     THEN SPVLIST.NEXT->SPVLIST->NAMEV(LIST.HD); 0->TAG(LIST.HD);
CLOSE;
UNIQNAME(LIST.TL)
END



FUNCTION VARLSFN ITEM;
COMMENT 'LISTS VARIABLES OF A CLAUSE`;
IF ITEM.ISVAR AND NOT(MEMBER(ITEM,VARSLIST))
             THEN ITEM::VARSLIST->VARSLIST
ELSEIF ITEM.ISFUN THEN APPLIST(ARGLISTF(ITEM),VARLSFN)
ELSEIF ITEM.ISPRED THEN APPLIST(ARGLISTP(ITEM),VARLSFN)
ELSEIF ITEM.ISPAIR THEN APPLIST(ITEM,VARLSFN)
CLOSE;
END



FUNCTION GET N LIST;
COMMENT 'GETS THE NTH ITEM OF A LIST`;
LOOP: IF N=1 THEN LIST.HD
             ELSE LIST.TL->LIST; N-1->N; GOTO LOOP
      CLOSE
END;


LAMBDA INDEX LIST LISTLENGTH;
GET(LISTLENGTH-INDEX+1,LIST)
END(%NIL,0%)->CLS;
LAMBDA NEWITEM INDEX;
IF FROZVAL(2,CLS)+1=INDEX
  THEN NEWITEM::FROZVAL(1,CLS)->FROZVAL(1,CLS);
     FROZVAL(2,CLS)+1->FROZVAL(2,CLS)
  ELSE 'CLS UPDATE ERROR`=>
CLOSE
END->UPDATER(CLS);




LAMBDA CLAUSE N;
CLAUSE->CLS(N);
[% "OK","NUMBER",N %].PR; NL(1);
N+1->FROZVAL(1,ADDCLAUSE);
UPDVARS(N+1);
END(%1%)->ADDCLAUSE;
COMMENT ADDCLAUSE ADDS A CLAUSE TO 'CLS`;



FUNCTION REMOVE N LIST;
COMMENT 'REMOVES NTH ITEM FROM A LIST`;
IF LIST.NULL THEN UNDEF
ELSEIF N=1 THEN LIST.TL
ELSE (LIST.HD)::REMOVE(N-1,LIST.TL)
CLOSE;
END
```

```
FUNCTION NCJ XS1 XS2;
COMMENT 'NON-CONSTRUCTIVE JOIN OF TWO LISTS';
VARS START; XS1->START;
IF XS1.NULL THEN XS2 EXIT;
LOOP: IF XS1.TL.NULL THEN XS2->XS1.TL; START EXIT;
XS1.TL->XS1;
GOTO LOOP;
END

FUNCTION RESOL LN1 CL1 LN2 CL2;
COMMENT 'RESOLVES TWO CLAUSES';
VARS L1 L2 RESULT TIMES;
COPYCL(CL1)->CL1; COPYCL(CL2)->CL2;
GET(LN1,CL1)->L1; GET(LN2,CL2)->L2;
NIL->DICT;
IF NOT(UNIFYRES(L1,L2)) THEN UNDEF EXIT;
NCJ(REMOVE(LN1,CL1),REMOVE(LN2,CL2))->RESULT;
LENGTH(DICT)->TIMES;
LOOP: TIMES-1->TIMES;
       SIGMA(RESULT)->RESULT;
       IF TIMES>0 THEN GOTO LOOP CLOSE;
PRUNE(RESULT);
NIL->VARSLIST;
VARLSFN(RESULT);
SUBTRSET(VARLIST,MAPLIST(VARSLIST,NAMEV))->SPVLIST;
UNIQNAME(VARSLIST);
RESULT;
END
```

```
FUNCTION RESOLVE CN1 LN1 CN2 LN2;
VARS ENT;
RESOL(LN1,CLS(CN1),LN2,CLS(CN2))->ENT;
IF ENT="UNDEF" THEN [SORRY - WONT RESOLVE].PR; NL(1);
ELSE ENT.PRR; ADDCLAUSE(ENT)
CLOSE;
END
```

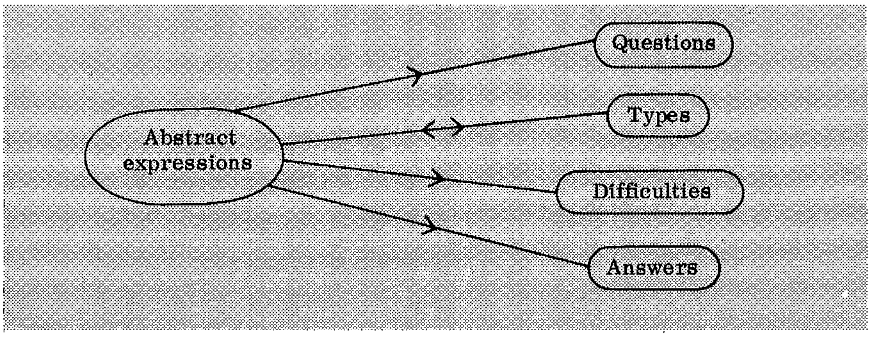
```
FUNCTION FACT LN1 LN2 CL;
COMMENT 'FACTORS A CLAUSE';
VARS L1 L2 TIMES;
COPYCL(CL)->CL;
GET(LN1,CL)->L1; GET(LN2,CL)->L2;
IF NOT(UNIFYFAC(L1,L2)) THEN UNDEF EXIT;
LENGTH(DICT)->TIMES;
LOOP: TIMES-1->TIMES;
SIGMA(CL)->CL;
  IF TIMES>0 THEN GOTO LOOP CLOSE;
PRUNE(CL)
END

FUNCTION FACTOR CLN LN1 LN2;
VARS ENT;
FACT(LN1,LN2,CLS(CLN))->ENT;
IF ENT="UNDEF" THEN [SORRY - WONT FACTOR].PR; NL(1);
ELSE ENT.PRR; ADDCLAUSE(ENT)
CLOSE;
END
```

sequence of questions can be interrupted in the usual way by typing the CONTROL & G character.

Typing SWAP before typing BEGIN reverses the question language and the answer language.

It is to be noted that the answer checking is done by direct matching and synonyms will not be recognized as being correct. A user will soon learn which particular word the program expects as the 'correct' meaning of a word.

¶*Method used.* The method used is simple. It gains a certain generality and conciseness by being based on some elementary principles of general algebra, roughly speaking the notion of a general evaluation mechanism (the extension of a mapping to a homomorphism). It is summarized in the following diagram.



Abstract expressions are generated randomly and 'evaluated' in four ways: to obtain a type, which must be the desired type and not undefined; to obtain a difficulty, which must be in the required range; to obtain a question; and to obtain an answer. In practice the 'evaluation' of type is inverted so that a type, for example, 'sentence' or 'number', is selected first and then an abstract expression is generated, which is sure to evaluate to this type. Roughly, the whole process is a generalization of the method used to generate well-formed expressions of a phrase structure grammar.

¶*Global variables and functions.* FENG, FGERM, FFINN, FSORT, FDIFF, FSTR, DSORT, ARITY, OMEGASET, LIST, LIST1, TEST1, TEST2, AASTRLAN, AANUMLAN, AAWORD, AADIFF, AAENG, AAGERM, PHIDIFF, PHISORT, PHIQUE, PHIANS, SAMEANS, PRANS, READANS, DMAX, DMIN, CORRECT, NAME, COUNT, R, ISVAR, NTUPLE, KDERIV, QUEST, ANSW, REP, ENGNPH, PRIMLIST, EXPD, EXPAND, TRANSLATE, ENGLISH, GERMAN, FINNISH, SUMS1, SUMS2. In addition all global variables in LIB NEW STRUCTURES and in LIB RANDOM.

¶*Store used.* The program uses a large amount of workspace and users are advised to have at least 25 blocks of store available before attempting to use this program.

¶*Examples of use*

```
: COMPILE(LIBRARY([LIB SUMS QUIZ]));

TO ENTER PROGRAM TYPE

BEGIN

: BEGIN
```

```
WHAT IS YOUR NAME:  JEROME

TYPE ANSWER TO THIS SUM

[10 × 8]
:80
GOOD
TYPE ANSWER TO THIS SUM

[7 + 4]
:  10
NO
TYPE YES IF YOU WANT TO TRY AGAIN ELSE TYPE NO: YES
OK: 11
THAT IS RIGHT
TYPE ANSWER TO THIS SUM
[9 * 1 + 1]
:  9
NO
TYPE YES IF YOU WANT TO TRY AGAIN ELSE TYPE NO : NO
THE RIGHT ANSWER IS 10
.
.
.
etc.
:COMPILE(LIBRARY([LIB LANGUAGE QUIZ]));

THERE ARE THREE LANGUAGES AVAILABLE FOR TRANSLATION
        VIZ. ENGLISH FINNISH AND GERMAN.

WHICH LANGUAGE DO YOU WISH TO TRANSLATE FROM : GERMAN

WHICH LANGUAGE DO YOU WISH TO TRANSLATE TO : ENGLISH

THERE WILL BE A SHORT PAUSE WHILE THE RELEVANT
        PROGRAMS ARE COMPILED

TO ENTER PROGRAM TYPE
BEGIN
PLEASE END ALL ANSWERS WITH A COMMA.

:  BEGIN

WHAT IS YOUR NAME:  JEROME,
TRANSLATE
[FUNF]
:  FIVE,
CORRECT
TRANSLATE

[DER GUTE MANN]
:  THE GOOD MEN,
NO
'TYPE YES IF YOU WANT TO TRY AGAIN ELSE NO':  NO
'THE RIGHT ANSWER IS' [THE GOOD MAN]
.
.
.
etc.
```

```
[QUIZZING MACHINE]


POPMESS([OPERATOR 'QUIZZING MACHINE IN USE. CHECK AUTHORISATION']);

ERASE->CUCHAROUT;

COMPILE(LIBRARY([LIB NEW STRUCTURES]));
COMPILE(LIBRARY([LIB RANDOM]));

VARS PHISORT FSORT AASORT ARITY ISVAR AAWORD OMEGASET LIST1 PHIDIFF R COUNT
     NAME CORRECT DMAX DMIN  PHIQUE PHIANS DSORT SAMEANS PRANS
     PROGRESS INV PRIM AGREES READANS  OPERATION 2 (>>> <<<);

LAMBDA U; IF U.ISWORD OR U.ISINTEGER THEN TRUE ELSE FALSE CLOSE END->ISVAR;



FUNCTION EXTD E F AA;
  FUNCTION PHI E; VARS AAOMEGA;
    IF E.ISVAR THEN F(E)
      ELSE E.DATAWORD.AA->AAOMEGA; AAOMEGA(APPLIST(E.DATALIST,PHI))
    CLOSE
  END
  PHI(E);
END



FUNCTION LISTN N;
  IF N=0 THEN NIL ELSE 0::LISTN(N-1); CLOSE;
END



FUNCTION WORDALG OMEGASET;
  VARS AAWORD; .NEWNNARRAY->AAWORD;
  APPLIST(OMEGASET,LAMBDA OMEGA;
                     RECORDFNS(OMEGA,LISTN(ARITY(OMEGA)));
                     APPLIST(LISTN(ARITY(OMEGA)+1),LAMBDA X; .ERASE; END);
                       ->AAWORD(OMEGA);
                   END);
  AAWORD
END



FUNCTION CHOOSE LIST;
  VARS X N;0->N;
  INTOF(RANDOM()*LENGTH(LIST))->X;
LOOP: IF N=X THEN LIST.HD EXIT;
  N+1->N; LIST.TL->LIST; GOTO LOOP
END



VARS NTUPLE LIST;
  COMMENT THESE VARIABLES ARE ACTUALLY LOCAL TO DERIV BUT HAVE
    BEEN TAKEN OUT TO SAVE STACK SPACE;

VARS KDERIV; 0.67->KDERIV;



FUNCTION DERIV S; VARS OMEGA;
  IF PRIM(S) AND DSORT(S)=UNDEF  OR .RANDOM<KDERIV
     THEN CHOOSE(INV(FSORT,S))
    ELSE CHOOSE(DSORT(S))->LIST;
      LIST.HD->OMEGA;  LIST.TL.HD->NTUPLE;
      APPLY(APPLIST(NTUPLE,DERIV),AAWORD(OMEGA));
  CLOSE
END
```

```
FUNCTION RECPR X;
  IF X.ISVAR THEN PR(X); SP(1);
LOOP: IF LIST1.HD=100 THEN EXIT;
    LIST1.HD-1->LIST1.HD;
    IF LIST1.HD=0 THEN PR(")");
      LIST1.TL->LIST1;  GOTO LOOP
    CLOSE;
    ELSE PR(DATAWORD(X));  PR("(");
      ARITY(DATAWORD(X))::LIST1->LIST1;
      APPLIST(DATALIST(X),RECPR);
  CLOSE;
END;


FUNCTION INV NNARRAY CPT;
  VARS LIST; []->LIST;
  FROZVAL(1,NNARRAY)->ASS;
LOOP: IF ASSNULL(ASS) THEN LIST EXIT;
  IF EQUAL( ASSCPT(ASS),CPT) THEN ASSSUB(ASS)::LIST->LIST; CLOSE;
  ASSTL(ASS)->ASS;
  GOTO LOOP
END

FUNCTION ASKNAME;
  VARS INPUT; CHARIN.INCHARITEM->INPUT;
    2.NL;
      PR('WHAT IS YOUR NAME');
      INPUT()->NAME;
    2.NL;
END

FUNCTION READNTW=>X; VARS INPUT;
  CHARIN.INCHARITEM->INPUT;
  INPUT()->X;
  IF X="-" THEN - INPUT()->X EXIT;
  IF X="TRUE" THEN TRUE->X
    ELSEIF X="FALSE" THEN FALSE->X
  CLOSE
END;

FUNCTION READWL=>L; VARS INPUT X;
   CHARIN.INCHARITEM->INPUT; NIL->L;
LOOP: INPUT()->X;
  IF X="," THEN REV(L)->L EXIT;
  IF X="-" THEN INPUT()->X; CLOSE;
  X::L->L; GOTO LOOP
END

EQUAL->SAMEANS; PR->PRANS;
NONOP > ->NONOP >>>;
NONOP < ->NONOP <<<;
CHAROUT->CUCHAROUT;

FUNCTION PROBLEM SORT MESSAGE;
  VARS E QUESTION ANSWER ERRORSLIST DIFF INPUT;
  0->COUNT; 0->CORRECT; NIL->ERRORSLIST;
  .ASKNAME;
LOOP: COUNT+1->COUNT;
     IF NOT(ERRORSLIST.NULL) AND .RANDOM<0.6
       THEN ERRORSLIST.DEST->ERRORSLIST; .DEST->ANSWER->QUESTION;
       ELSE L0: DERIV(SORT)->E;
                E.PHIDIFF->DIFF;
             IF DIFF>>>DMAX OR DIFF<<<DMIN THEN GOTO L0 CLOSE;
         E.PHIQUE->QUESTION;
         E.PHIANS->ANSWER;
     CLOSE;
  1.NL; MESSAGE.PR; 2.NL;  QUESTION.PR; 1.NL;
LANS: CHARIN.INCHARITEM->INPUT;
  IF AGREES(.READANS,ANSWER)
       THEN CORRECT+1->CORRECT;  .PROGRESS;
      ELSE 1.NL; 'TYPE YES IF YOU WANT TO TRY AGAIN ELSE TYPE NO'.PR;
             IF .INPUT ="YES" THEN "OK".PR;  GOTO LANS CLOSE;
        ERRORSLIST<>[%QUESTION::ANSWER%]->ERRORSLIST;
      PR('THE RIGHT ANSWER IS');  ANSWER.PRANS;
      CLOSE;
    GOTO LOOP
END;
```

```
FUNCTION AGREES OFFER ANSWER=>OK;
  SAMEANS(OFFER,ANSWER)->OK;
  IF OK THEN DMAX/R->DMAX;
             PR(CHOOSE([GOOD CORRECT [THAT IS RIGHT]]));
    ELSE DMAX*R->DMAX;
       PR("NO")
  CLOSE;
  DMAX -7->DMIN
END;

FUNCTION PROGRESS ;
  VARS REM;
  CORRECT//5->REM->REM;
  IF REM=0 THEN 1.NL;
     PR(CHOOSE([[WELL DONE][VERY GOOD][GOOD SHOW]])
        <>(NAME::[,YOU HAVE]<>(CORRECT::[RIGHT ANSWERS])))
  CLOSE;
  CORRECT//20->REM->REM;
    IF REM=0 THEN 2.NL;
      'GO TO THE TOP OF THE CLASS`.PR; 2.NL;
    CLOSE
END;



INTOF(HD(.POPDATE)*100)->RANSEED;
[100]->LIST1;
0.95->R;
12->DMAX;
5->DMIN;


[LANGUAGE QUIZ]

VARS QUEST ANSW REP;


COMPILE(LIBRARY([LIB QUIZZING MACHINE]));
COMPILE(LIBRARY([LIB LANGUAGE GENERAL]));
COMPILE(LIBRARY([LIB LANGUAGE SORT]));
2.NL;

PR('THERE ARE THREE LANGUAGES AVAILABLE FOR TRANSLATION
    VIZ. ENGLISH FINNISH AND GERMAN.
WHICH LANGUAGE DO YOU WISH TO TRANSLATE FROM`);
CHARIN.INCHARITEM->REP;
.REP->QUEST;
2.NL;
PR('WHICH LANGUAGE DO YOU WISH TO TRANSLATE TO`);
CHARIN.INCHARITEM-> REP;
.REP->ANSW;
; 2.NL;

PR('THERE WILL BE A SHORT PAUSE WHILE THE RELEVANT PROGRAMS ARE
    COMPILED`);  2.NL;

COMPILE(LIBRARY([%"LIB",QUEST%]));
COMPILE(LIBRARY([%"LIB",ANSW%]));


VALOF(QUEST)->PHIQUE;
VALOF(ANSW)->PHIANS;


MACRO BEGIN;  2.NL;
  12->DMAX;
  5->DMIN;
  .TRANSLATE;
END;

PR('
     TO ENTER PROGRAM TYPE
            BEGIN

  PLEASE END ALL ANSWERS WITH A COMMA.
`);
2.NL;
```

```
[LANGUAGE GENERAL]

VARS EXPD EXPAND TRANSLATE;

.NEWNNARRAY->EXPD;
.NEWNNARRAY ->ARITY;


[NOUN AN SING NOM]->EXPD("NASN");
[NOUN AN PL NOM]->EXPD("NAPN");
[NOUN AN SING ACC]->EXPD("NASA");
[NOUN AN PL ACC]->EXPD("NAPA");
[NOUN INAN SING NOM]->EXPD("NISN");
[NOUN INAN PL NOM]->EXPD("NIPN");
[NOUN INAN SING ACC]->EXPD("NISA");
[NOUN INAN PL ACC]->EXPD("NIPA");

[ADJ AN SING NOM]->EXPD("AASN");
[ADJ AN PL NOM]->EXPD("AAPN");
[ADJ AN SING ACC]->EXPD("AASA");
[ADJ AN PL ACC]->EXPD("AAPA");
[ADJ INAN SING NOM]->EXPD("AISN");
[ADJ INAN PL NOM]->EXPD("AIPN");
[ADJ INAN SING ACC]->EXPD("AISA");
[ADJ INAN PL ACC]->EXPD("AIPA");

[VERB AN SING]->EXPD("VAS");
[VERB AN PL]->EXPD("VAP");
"REMARK"->EXPD("REM");

VARS OPERATION 2 TONARR;


FUNCTION TONARR L A; VARS X Y;
LOOP: IF L.NULL THEN EXIT;
  L.NEXT->L->Y; L.NEXT->L->X;
  IF EXPAND THEN EXPD(Y)->Y CLOSE; Y->A(X); GOTO LOOP
END



[MKREMARK  MKNONPH1 MKNONPH2 MKNONPH3 MKNONPH4
   MKACNPH1 MKACNPH2 MKACNPH3 MKACNPH4
    MKSENT1 MKSENT2]->OMEGASET;
2->ARITY("MKNONPH1");
2->ARITY("MKNONPH2");
2->ARITY("MKNONPH3");
2->ARITY("MKNONPH4");
1->ARITY("MKREMARK");
2->ARITY("MKACNPH1");
2->ARITY("MKACNPH2");
2->ARITY("MKACNPH3");
2->ARITY("MKACNPH4");

3->ARITY("MKSENT1");
3->ARITY("MKSENT2");

WORDALG(OMEGASET)->AAWORD;
READWL->READANS;


FUNCTION EXPRDIFF E REM  NPH  SENT;
  IF E.DATAWORD ="MKREMARK" THEN
    IF DMAX<<<REM  THEN DMAX-1->REM;
    CLOSE;  REM;
    ELSEIF E.DATAWORD ="MKNONPH1" THEN NPH
    ELSE
    IF DMIN>>>SENT THEN DMIN+1->SENT;
    CLOSE;  SENT;
   CLOSE
END;

EXPRDIFF(% 8, 12, 16 %)->PHIDIFF;
PROBLEM(%"EXPRESSION",'TRANSLATE`%)->TRANSLATE;

MACRO SWAP;
  PHIANS,PHIQUE->PHIANS->PHIQUE
END;
```

```
[LANGUAGE SORT]

VARS DSORT PRIMLIST;
.NEWNNARRAY->FSORT;
.NEWNNARRAY->AASORT;
.NEWNNARRAY->DSORT;

[[MKNONPH1 [[ADJ AN SING NOM] [NOUN AN SING NOM]]]]->DSORT("NONPH1");
    [[MKNONPH2 [[ADJ AN PL NOM][NOUN AN PL NOM]]]]->DSORT("NONPH2");
     [[MKNONPH3 [[ADJ INAN SING NOM] [NOUN INAN SING NOM]]]]->DSORT("NONPH3");
    [[MKNONPH4 [[ADJ INAN PL NOM] [NOUN INAN PL NOM]]]]->DSORT("NONPH4");

[[MKACNPH1 [[ADJ AN SING ACC] [NOUN AN SING ACC]]]]->DSORT("ACNPH1");
[[MKACNPH2 [[ADJ AN PL ACC] [NOUN AN PL ACC]]]]->DSORT("ACNPH2");
[[MKACNPH3[[ADJ INAN SING ACC][NOUN INAN SING ACC]]]]->DSORT("ACNPH3");
[[MKACNPH4 [[ADJ INAN PL ACC] [NOUN INAN PL ACC]]]]->DSORT("ACNPH4");

[[MKREMARK [REMARK]]
   [MKNONPH1 [[ADJ AN SING NOM][NOUN AN SING NOM]]]
   [MKNONPH2 [[ADJ AN PL NOM][NOUN AN PL NOM]]]
   [MKSENT1 [NONPH1 [VERB AN SING] ACNPH4]]
  [MKSENT1 [NONPH1 [VERB AN SING] ACNPH2]]
   [MKSENT1 [NONPH2 [VERB AN PL] ACNPH1]]
   [MKSENT1 [NONPH2 [VERB AN PL] ACNPH3]]]->DSORT("EXPRESSION");

[[NOUN AN SING NOM][NOUN AN PL NOM][NOUN INAN SING NOM]
        [NOUN INAN PL NOM][NOUN AN SING ACC][NOUN AN PL ACC]
       [NOUN INAN SING ACC][NOUN INAN PL ACC][ADJ AN SING NOM]
     [ADJ AN PL NOM][ADJ INAN SING NOM][ADJ INAN PL NOM]
     [ADJ AN SING ACC][ADJ AN PL ACC][ADJ INAN SING ACC]
     [ADJ INAN PL ACC][VERB AN SING][VERB AN PL][VERB INAN SING]
     [VERB INAN PL]REMARK]->PRIMLIST;

FUNCTION MEMBER  ITEM LIST EQUIV;
 LO: IF LIST.NULL THEN FALSE EXIT;
     IF EQUIV(ITEM, LIST.HD) THEN TRUE EXIT;
   LIST.TL->LIST;
  GOTO LO
END;

FUNCTION PRIM X;
  IF MEMBER(X,PRIMLIST,EQUAL) THEN TRUE ELSE FALSE CLOSE
END;

TRUE->EXPAND;

[NASN 1 NASN 21 NAPN 2 NAPN 22 NASA 3 NASA 23 NAPA 4 NAPA 24
 NISN 5 NIPN 6 NISA 7 NIPA 8 AASN 9 AASN 29 AAPN 10 AAPN 30
AASA 11 AASA 31 AAPA 12 AAPA 32 AISN 13 AIPN 14 AISA 15 AIPA 16
VAS 17 VAS 37 VAP 18 VAP 38 REM 40 REM 41 REM 42 REM 43 REM 44
REM 45 REM 46 REM 47] TONARR FSORT;

FALSE->EXPAND;




[ENGLISH]

VARS FENG AAENG ENGNPH ENGLISH;
.NEWNNARRAY->FENG;
.NEWNNARRAY->AAENG;


LAMBDA  R; IF R.ATOM THEN [%R%] ELSE R CLOSE  END->AAENG("MKREMARK");
LAMBDA  A N;[THE]<>[%A%]<>[%N%] END->ENGNPH;
ENGNPH->AAENG("MKNONPH1");
ENGNPH->AAENG("MKNONPH2");
ENGNPH->AAENG("MKNONPH3");
ENGNPH->AAENG("MKNONPH4");
ENGNPH->AAENG("MKACNPH1");
ENGNPH->AAENG("MKACNPH2");
ENGNPH->AAENG("MKACNPH3");
ENGNPH ->AAENG("MKACNPH4");

LAMBDA NP V ACNP; NP<>[%V%]<>ACNP END->AAENG("MKSENT1");

FALSE->EXPAND;
```

```
[MAN 1 MEN 2 MAN 3 MEN 4  BOY 21 BOYS 22 BOY 23 BOYS 24
 TABLE 5 TABLES 6 TABLE 7 TABLES 8 GOOD 9 GOOD 10 GOOD 11 GOOD 12
BIG 29/ BIG 30 BIG 31 BIG 32 SMALL 13 SMALL 14 SMALL 15 SMALL 16
SEES 17 SEE 18 CARRIES 37 CARRY 38
[GOOD MORNING] 40 [GOOD DAY] 41 GOODBYE 42
ONE 43 TWO 44 THREE 45 FOUR 46 FIVE 47] TONARR FENG;

EXTD(%FENG,AAENG%)->ENGLISH;




[FINNISH]

ERASE->CUCHAROUT;
COMPILE(LIBRARY([LIB ENGLISH]));

CHAROUT->CUCHAROUT;
VARS FFINN FINNISH AAENG;

.NEWNNARRAR->FFINN;

FUNCTION AAFINN X;
  IF X = "MKSENT1" OR X="MKREMARK"
 THEN AAENG(X) ELSE LAMBDA A N; [%A%]<>[%N%] END
  CLOSE
END;

[MIES 1 MIEHET 2 MIEHEN 3 MIEHET 4
KILTTI 9 KILTIT 10 KILTIN 11 KILTIT 12
 NAKEE 17 NAKEVAT 18
 POYTA 5 POYDAT 6 POYDAN 7 POYDAT 8
 PIENI 13 PIENET 14 PIENEN 15 PIENET 16
POIKA 21 POJAT 22 POJAN 23 POJAT 24
ISO 29 ISOT 30 ISON 31 ISOT 32
KANTAA 37 KANTAVAT 38
HUOMENTA 40 PAIVAA 41 NAKEMIIN 42
 YKSI 43 KAKSI 44 KOLME 45 NELJA 46 VIISI 47] TONARR FFINN;

EXTD(%FFINN,AAFINN%)->FINNISH;




[GERMAN]

VARS FGERM AAGERM GERMAN;
.NEWNNARRAY->FGERM;
.NEWNNARRAY->AAGERM;

LAMBDA R; IF R.ATOM THEN [%R%] ELSE R CLOSE  END->AAGERM("MKREMARK");
LAMBDA   A N;[DER]<>[%A%]<>[%N%] END->AAGERM("MKNONPH1");
LAMBDA A N;[DIE]<>[%A%]<>[%N%] END->AAGERM("MKNONPH2");
LAMBDA A N;[DER]<>[%A%]<>[%N%] END->AAGERM("MKNONPH3");
LAMBDA A N;[DIE]<>[%A%]<>[%N%] END->AAGERM("MKNONPH4");
LAMBDA A N;[DEN]<>[%A%]<>[%N%] END->AAGERM("MKACNPH1");
LAMBDA A N;[DIE]<>[%A%]<>[%N%] END->AAGERM("MKACNPH2");
LAMBDA A N;[DEN]<>[%A%]<>[%N%] END->AAGERM("MKACNPH3");
LAMBDA A N;[DIE]<>[%A%]<>[%N%] END->AAGERM("MKACNPH4");

FALSE->EXPAND;

[MANN 1 MANNER 2 MANN 3 MANNER 4 KNABE 21 KNABEN 22 KNABEN 23 KNABEN 24
TISCH 5 TISCHE 6 TISCH 7 TISCHE 8
GUTE 9 GUTEN 10 GUTEN 11 GUTEN 12
GROSSE 29 GROSSEN 30 GROSSEN 31 GROSSEN 32

KLEINE 13 KLEINEN 14 KLEINEN 15 KLEINEN 16
SIEHT 17 SEHEN 18 TRAGT 37 TRAGEN 38
[GUTEN MORGEN] 40 [GUTEN TAG] 41 [AUF WIEDER SEHEN] 42
EINS 43 ZWEI 44 DREI 45 VIER 46 FUNF 47] TONARR FGERM;


LAMBDA NP V ACNP; NP<>[%V%]<>ACNP END->AAGERM("MKSENT1");
EXTD(%FGERM,AAGERM%)->GERMAN;
```

```
[SUMS]

VARS  AASTRLAN AANUMLAN FDIFF AADIFF PHIDIFF FSTR TEST1 TEST2;

.NEWNNARRAY->FSORT;
.NEWNNARRAY->AASORT;
.NEWNNARRAY->ARITY;
.NEWNNARRAY->AASTRLAN;
.NEWNNARRAY->AANUMLAN;
.NEWNNARRAY->DSORT;
.NEWNNARRAY->FDIFF;
.NEWNNARRAY->AADIFF;

[PLUS MINUS TIMES GRTHAN NOT]->OMEGASET;;



"NUM"->FSORT(0);
"NUM"->FSORT(1);
"NUM"->FSORT(2);
"NUM"->FSORT(3);
"NUM"->FSORT(4);
"NUM"->FSORT(5);
"NUM"->FSORT(6);
"NUM"->FSORT(7);
"NUM"->FSORT(8);
"NUM"->FSORT(9);
"NUM"->FSORT(10);



2->ARITY("PLUS");
2->ARITY("MINUS");
2->ARITY("TIMES");
2->ARITY("GRTHAN");
1->ARITY("NOT");

FUNCTION TEST U V X Y R;
  IF U=X AND V=Y THEN R ELSE UNDEF CLOSE
END;

TEST(%"NUM","NUM","NUM"%)->TEST1;
TEST(%"NUM","NUM","TVAL"%)->TEST2;

TEST1->AASORT("PLUS");
TEST1->AASORT("MINUS");
TEST1->AASORT("TIMES");
TEST2->AASORT("GRTHAN");

LAMBDA U; IF U="TVAL" THEN "TVAL" ELSE UNDEF CLOSE END->AASORT("NOT");



LAMBDA U V;
  IF U.ATOM THEN [%U%] ELSE U
  CLOSE;  <>[+]<> IF V.ATOM THEN [%V%] ELSE [%"("%]<> V<>[%")"%] CLOSE
END->AASTRLAN("PLUS");

LAMBDA U V;
  IF U.ATOM THEN [%U%]  ELSE U
  CLOSE  <>[-]<> IF V.ATOM THEN [%V%] ELSE
    [%"("%]<>V<>[%")"%] CLOSE
END->AASTRLAN("MINUS");

LAMBDA  U V;
  IF U.ATOM THEN [%U%] ELSE[%"("%]<> U<>[%")"%]
  CLOSE  <>[*]<> IF V.ATOM THEN [%V%] ELSE
     [%"("%]<>V<>[%")"%] CLOSE
END->AASTRLAN("TIMES");

LAMBDA U V;
  IF U.ATOM THEN   [%U%] ELSE U
  C  <>[>]<> IF V.ATOM THEN [%V%] ELSE V CLOSE;
END->AASTRLAN("GRTHAN");

LAMBDA U; [NOT]<> IF U.ATOM THEN [%U%] ELSE U CLOSE;
END->AASTRLAN("NOT");
```

```
LAMBDA U V; U+V END->AANUMLAN("PLUS");
LAMBDA U V; U-V END->AANUMLAN("MINUS");
LAMBDA U V; U*V END->AANUMLAN("TIMES");
LAMBDA U V; U>V END->AANUMLAN("GRTHAN");
LAMBDA U; NOT(U) END->AANUMLAN("NOT");

-2->FDIFF(0);
-1->FDIFF(1);
-1->FDIFF(2);
-2->FDIFF(3);
-3->FDIFF(4);
-3->FDIFF(5);
-5->FDIFF(6);
-6->FDIFF(7);
-6->FDIFF(8);
-6->FDIFF(9);
-2->FDIFF(10);

FUNCTION MOD X;
  IF X<0 THEN -X  ELSE X CLOSE;
END;


LAMBDA U V; U.MOD + 1 + V.MOD END->AADIFF("PLUS");
LAMBDA U V; U.MOD + 2 + V.MOD END->AADIFF("MINUS");
LAMBDA U V; U.MOD + 4 + V.MOD END->AADIFF("TIMES");
LAMBDA U V; U.MOD + 3 + V.MOD END->AADIFF("GRTHAN");
LAMBDA U; U.MOD + 10 END->AADIFF("NOT");

FUNCTION PRIM X; X="NUM" END;


[[GRTHAN [NUM NUM]][NOT [TVAL]]]->DSORT("TVAL");
[[PLUS [NUM NUM]][MINUS [NUM NUM]][TIMES [NUM NUM ]]]->DSORT("NUM");

LAMBDA X; X END->FSTR;


EXTD(%FSORT,AASORT%)->PHISORT;
EXTD(%FSTR,AASTRLAN%)->PHIQUE;
EXTD(%FSTR,AANUMLAN%)->PHIANS;
EXTD(%FDIFF,AADIFF%)->PHIDIFF;


WORDALG(OMEGASET)->AAWORD;

VARS SUMS1 SUMS2;
PROBLEM(%"NUM",'TYPE ANSWER TO THIS SUM`%)->SUMS1;
PROBLEM(%"TVAL",'TYPE ANSWER TO THIS SUM`%)->SUMS2;

MACRO BEGIN;
12->DMAX;
5->DMIN;
.SUMS1;
END;

READNTW->READANS;

PR('

TO ENTER PROGRAM TYPE

BEGIN
`); 2.NL;
```