stichting
mathematisch
centrum

MC

AFDELING INFORMATICA
(DEPARTMENT OF COMPUTER SCIENCE)

IW 122/79 NOVEMBER

D. GRUNE

THE REVISED MC ALGOL 68 TEST SET

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O).

1980 Mathematics subject classification: 68-04

ACM-Computing Reviews-category: 4.6, 4.12

The Revised MC ALGOL 68 Test Set

by

Dick Grune (ed.)

ABSTRACT

The Revised MC ALGOL 68 Test (Version 3.0) Set is comprised of 190 ALGOL 68 programs covering the full language with the exception of bulk I/O. Most programs have been run on at least two ALGOL 68 systems.

KEYWORDS & PHRASES: ALGOL 68, compilers, debugging.

The following statement has been authorized by IFIP Working Group 2.1:

> This ALGOL68 Test Set has been reviewed by IFIP Working Group 2.1, which considers it as a valuable means of testing implementations of ALGOL68.

0. IMPORTANT NOTES

0.1. On ALGOL 68

This is a test set, not a collection of programs to learn ALGOL 68 from: many of the programs are pathological and in no way representative for ALGOL 68 programming. (But see chapter 4.)

Readers not thoroughly familiar with ALGOL 68 could read (in order of increasing difficulty):

- A.S. Tanenbaum, A Tutorial on ALGOL 68, Computing Surveys, 8, p155-190, 1976.
- F.G. Pagan, A Practical Guide to ALGOL 68, John Wiley & Sons, London, 1976.
- S.G. van der Meulen and P. Kühling, Programmieren in ALGOL 68, Walter de Gruyter, Berlin, 1974 (in German).
- C.H. Lindsey and S.G. van der Meulen, Informal Introduction to ALGOL 68 Revised, North Holland Publ. Comp., Amsterdam, 1977.
- A. van Wijngaarden et al., editors, Revised Report on the Algorithmic Language ALGOL 68, Acta Informatica, 5, 1-236, 1975; MC Tract 50, Mathematical Centre, 1976; SIGPLAN Notices, 12, 5, p1-70, 1977.

0.2. On ALGOL 68 S

This is not an ALGOL 68 S test set ("ALGOL 68 S" stands for the IFIP sublanguage [1]). It explicitly explores all odd corners of ALGOL 68, which makes it unsuitable as a measuring tool for ALGOL 68 S: three quarters of the programs presented here will not run even on a perfect ALGOL 68 S implementation. For adapting the programs to ALGOL 68 S, see chapter 8.

1. HISTORY

The test set here described grew out of a number of needs and desires. In the years after the publication of the original report, many of us began slowly to gain some understanding of the language and started to write small programs like the Towers of Hanoi, to get a feel of the power and conciseness of the language, or, after having studied coercions, someone would write a five-line program, containing the most complicated soft-balance he could dream up. But we had no compiler then and these programs remained desk exercises.

In 1973 the ALGOL 68 interpreter by L. Ammeraal [2] became operative, for a subset, it is true, but for a convenient subset. We immediately grasped the opportunity to use it for testing proposed rewritings of some transput sections like "whole", "fixed" and "float".

At the same time the need arose for test programs to test this interpreter. Now the situation would have been completely different if we had been concerned with a FORTRAN test set. For a well-established language like FORTRAN, a simple request at the bill board would yield a dozen or so solid, non-trivial, well-running programs, and a few words at coffee breaks would set some seasoned FORTRAN experts into motion to produce a collection of the most vicious detail tests imaginable. For ALGOL 68, however, both the well-running programs and the seasoned experts were lacking. So the compilers and the test set have grown in parallel and the test set has been revised, overhauled and re-edited several times, as the details of the Revised Report became clearer and the Control Data implementation evolved.

In 1972 Control Data Corporation Holland started the construction of a commercial full implementation of ALGOL 68 [3], to fulfil a contract with SARA (Stichting Academisch Rekencentrum Amsterdam, Amsterdam Universities Computing Centre) and ACCU (Academisch Computer Centrum Utrecht). Quality control on behalf of SARA was delegated to the Mathematical Centre, and this emphasized the need for a high-quality, multi-aspect test set.

With the advent of working but imperfect compilers another effect set in. To many programmers the presence of a large-size new system is a challenge, and programs began to come in that arose from questions like "Let's see what happens if I jump out of a format", or "Let's see if I can derail the parser". And since by mutual agreement the test set for the Control Data implementation would contain all programs on which it was still failing (!), these programs were welcomed by the manufacturer and me alike.

The test set was used extensively in the acceptance procedure of the Control Data ALGOL 68 Compiler. It was published as a Mathematical Centre Report [4] in December 1976. Since then many copies have found their way to (potential) ALGOL 68 implementers. Reactions came in and were incorporated into the test set. When the IFIP-endorsed Standard Hardware Representation [5] became available, the test set was made to conform to it. In April 1979 Working Group 2.1 of IFIP accepted the Formal Resolution presented above.

2. COMPLETENESS

The test set is not complete, in two respects. First there is of course that such a product is never complete: there is no exhaustive testing and one cannot cater for every contingency. Trivial as this may sound, I want to say a few more words on it. At least one quarter of the errors uncovered by the test set in the Control Data ALGOL 68 compiler were accidental discoveries. One such discovery illustrates my point so well that I will give it here: during straightening for output the internal length of an ob-

ject was calculated incorrectly if it was a union that originated from union-uniting. By pure accident the test set contains such an oddity (simple tests 8), and the error was uncovered.

The question is how much this affects the test set's efficacy, also in view of the fact that the test set is known beforehand. In my opinion, if a compiler processes the test set well and works well on the daily stream of average programs, it is a very good compiler. Through its unusual complexity, the test set will uncover most incorrect short-cuts, and the constant use of simple features will prevent the compiler from being too much tuned to the test set. I may have to make an exception for a heavily optimizing compiler. The present test set may be less effective there since such a compiler would often decide not to do any optimization at all: a special test set is needed based on knowledge of the optimization techniques.

The second respect in which it is not complete is that some aspects of the language are under-represented (e.g. SHORT and LONG values), or are not represented at all (bulk I/O). These defects could be remedied but only at the expense of serious postponement of publication.

Since the test set now conforms to the Standard Hardware Representation, there are no programs which test 'not-worthy' symbols like \, ⊥ or ~. If your installation has them, existing programs can easily be edited.

3. CONTENTS

Version 3.0 of the test set contains 190 programs, in part correct ones, in part intentionally incorrect ones, distributed over the following categories:

- null programs (9 programs)
  various forms of empty enclosed-clauses.

- simple tests (16 programs)
  contains almost all features of the language used in simple ways.

- declarers (7 programs)
  tests declarers and their well-formedness.

- mode equivalence (6 programs)

- operators (16 programs)
  tests operator-declarations and calls, including the UPB and LWB operators.

- identification (12 programs)
  tests identification of identifiers, mode-indications and operators, including LWB and UPB.

- clauses (9 programs)
  IF-clauses, CASE-clauses, loops, displays, vacuums.

- coercions (14 programs)

- identity relations (2 programs)

- stowed values (8 programs)
  tests selection and slicing, both of structures and rows.

- flexibility/transiency (4 programs)

- generators/garbage collection (4 programs)
  includes some timing of various operations.

- scope checking (10 programs)
  scope of generators, routine texts and formats.

- jumps (5 programs)

- parallel processing (6 programs)

- simple I/O (12 programs)
  tests some transput features in depth. The tests are simple in that they are mainly restricted to the file "stand out".

- standard environment (3 programs)
  these programs contain at least one applied-occurrence of every mode-indication, operator, identifier and field-selector defined in the standard environment.

- syntax errors (8 programs)
  contains elaborate and realistic attempts to derail the parser.

- miscellaneous misery (7 programs)
  various odds and ends that would not fit in other categories.

- numerical (13 programs)
  tests the mathematical functions and REAL arithmetic.

- application programs (19 programs)
  real-world though somewhat academic programs intended to run and yield sensible output, meanwhile testing many daily features in width rather than in depth.

4. READABILITY

Since most readers will probably read large parts of the test set before feeding it to an ALGOL 68 compiler, some words on the readability of the test set to humans seem in order.

Many of the programs are pathological and should not be considered as representative of ALGOL 68 programming style. With this in mind, almost all programs are worth while reading, some as puzzles, some for the good programming features they contain, some for their not widely known programming techniques and a few for their good style.

In my opinion the following programs are among the more interesting:

- simple tests 11
- declarers 3
- mode equivalence 2 (for a check to see if the compiler regards two modes as equivalent)
- jumps 5
- parallel processing 1 and 2
- numerical 3
- application 4, 5, 7, 10, 11, 16 and 17.

5. IMPLEMENTATION-INDEPENDENCE

A conscious effort has been made to keep the test set implementation-independent, facilitated by the fact that the Control Data implementation is almost a strict implementation. In 1978 a further thorough check was provided by the "Full Language ALGOL 68 Checkout Compiler" (FLACC) [6] of Chion Corporation.

I have attempted to remove all traces of Control Data influence: the only CHANNELs used are 'stand in channel', 'stand out channel' and 'stand back channel'; 'establish' and 'open' are used in the appropriate way, and so are LOC and HEAP. Since the test set now conforms to the Standard Hardware Representation, the question of operators not representable in Control Data Display Code does no longer arise. There are, however, no style-i-brackets.

Although most of this set has now been run on at least two systems, there must still be implementation-dependencies that I am not aware of.

6. RELIABILITY

All programs have been processed by the Control Data ALGOL 68 compiler. Most of them have also been processed by the FLACC compiler, including all those that are incompatible with the present version of the Control Data ALGOL 68 compiler. This has lead to a number of corrections, in particular in the realm of scope-checking and flexibility.

Although it is implicit in the above, I want to state explicitly that the programs in the category "parallel processing" have been tested and perform correctly.

7. HINTS FOR RUNNING THE PROGRAMS

Most programs are of less than average size and should run in default time and default space.

I hope that all programs are well enough commented for the reader to find out what they are supposed to do. Most programs are intended to run and print easily checkable output, either by first printing an example of the output or by printing a simple pattern.

A smaller number is intended to test the parser in its error-detecting capacity. In my opinion a good compiler will never crash and will yield enough readable information to allow the user to correct a large part of the errors (but perhaps not all).

Some programs will run, produce some output and then cause a run-time error. In a good system the error message should be understandable, in some symbolic form, and information printed previously should not be lost.

The following programs need special consideration:

- simple tests 16
  this program is its own input.

- operators 9, 10 and 11
  these contain all non-bold operators available in the Standard Hardware Representation. Each implementation should add their own 'not-worthy' operators.

- operator 13
  the ##= should be replaced by ≠-characters, if available (see note below on standard environment 1).

- numerical 4, applications 5 and 19
  these programs contain timing measurements, based on the presence of a 'PROC REAL clock' yielding the number of seconds elapsed since a fixed moment (e.g. begin of job or 68/01/01, 00:00:00).

- generators & garbage collection
  it may be necessary to run the programs in this category in a carefully estimated amount of space, so as to activate the garbage collector without getting stuck for lack of memory. Besides, these programs use some "user environment enquiries" for reporting about the garbage collector.

- simple I/O 1
  this program produces, in addition to the normal output file, a file ("ti") which is then read and checked by the program. It might be useful to have an independent look at the file afterwards.

- simple I/O 12
  the integer 'b' should be given a value in accordance to the accompanying comment.

- standard environment 1
  many operators from the standard environment contain symbols which do not exist in the Standard Hardware Representation, e.g., window-symbol. For a solution see the preface to the program.

- standard environment 2, application 15 and 19
  it may be necessary to adapt the parameters of 'open'.

- simple I/O 1, application 15 and 19
  it may be necessary to adapt the parameters of 'establish'.

- miscellaneous misery 2
  this program tries to fool the compiler into recognizing a format where there is none. It may be necessary to adapt it to the compiler being tested.

- syntax errors 5
  this is the empty program (not even a comment to tell what it is).

- application 15
  this program needs "stand in" and a file "program", as specified in the accompanying comment. It produces a garbled program that can be used to test the parser. In this sense the test set is open-ended.

- application 19
  sample input is given in a comment.

Many useful hints on testing compilers can be found in a paper by R.S. Scowen [7].

An independent test set of a completely different nature is available from B. Houssais, IRISA, Rennes, France.

8. ALGOL 68 S

About one quarter of the programs are correct ALGOL 68 S. Many of the remaining programs can be adapted to ALGOL 68 S by straightforward means:

- add declarations of certain standard prelude indicators (L int width, etc.),

- add GOTO-symbols,
- add lower-bounds of 1 in actual-rowers,
- reorder declarations,
- turn void-collateral-clauses into void-serial-clauses.

If a program cannot be adapted in this way, it is generally because it uses UNION, HEAP, FLEX, FORMAT or PAR; I see little point in elaborate attempts to adapt such programs.

9. AVAILABILITY

The test set can be obtained from:

Mathematical Centre,
Tweede Boerhaavestraat 49,
1091 AL Amsterdam,

in computer-readable form on 9-track ANSI Standard Labeled Tape, in ASCII. If necessary, other versions can be supplied, e.g. in EBCDIC or Control Data Display Code, or on paper-tape, 7-track magtape, ECMA cassette or in Braille.

The character set is the 'worthy character' set and programs are in point-style [5]. This publication uses upper-style for better readability.

10. CONTRIBUTERS AND ACKNOWLEDGEMENT

Contributions have come in from many places over a long period of time. I have tried to incorporate the programs as they were, only adding an occasional comment, reorganizing the output to facilitate checking, and, often, debugging them.

Since the origin of not all programs is known to me, the following list of contributers is probably incomplete.

Contributions were made by: J. Admiraal, H.J. Boom, C.J. Cheney, Th.J. Dekker, K. Gostelow, D. Grune, P.G. Hibbard, E. de Jong, J. Kok, T.J.G. Krijnen, L.G.L.T. Meertens, F. Teer, T. Toutenhoofd, M. Veldhorst, J.C. van Vliet, R. van Vliet, D.T. Winter, A. van Wijngaarden.

I thank all these people for their contributions without which this test set simply would not exist, and, knowing how much time and work it takes to make a test program, I deeply appreciate their effort.

11. LITERATURE

[1] P.G. Hibbard, A sublanguage of ALGOL 68, SIGPLAN Notices, 12, 5, p71-79, 1977.
[2] L. Ammeraal, An Interpreter for Simple ALGOL 68 Programs, IW 7/73, Mathematical Centre, Amsterdam, 1973.
[3] ALGOL 68 Version I Reference Manual, Control Data Services B.V., Rijswijk, Netherlands, 1976.
[4] D. Grune (ed.), The MC ALGOL 68 Test Set, IW 53/75, Mathematical Centre, Amsterdam, 1975.
[5] W.J. Hansen & H.J. Boom, The Report on the Standard Hardware Representation for ALGOL 68, SIGPLAN Notices, 12, 5, p80-87, 1977.
[6] FLACC User's Guide, Version 1.2, Chion Corporation, Box 4942, Edmonton, Alberta T6E 5G8, Canada, 1979.
[7] R.S. Scowen, The Diagnostic Facilities in ALGOL Compilers, NPL Report NAC52, July 1974, Teddington, England, 1974.

12. INFORMATION SHEET

```
abcdefghijklmnopqrstuvwxyz0123456789 "#$%'()*+,-./:;<=>@[]_|
ABCDEFGHIJKLMNOPQRSTUVWXYZ
!?
```

Contents: MC ALGOL 68 Test Set.
Version: 3.0.
Date: 13-03-1979.
Contact: Dick Grune,
Mathematisch Centrum,
Tweede Boerhaavestraat 49,
1091 AL Amsterdam,
The Netherlands.
Reference:
D. Grune, The Revised MC ALGOL 68 Test Set, IW 122/79,
Mathematical Centre, Amsterdam.
Modifications since version 2.3:
declarers 6, 7
operators 16
clauses 8, 9
stowed values 7, 8
flexibility/transiency 4
simple I/O 11, 12
numerical 2, 8, 9, 10, 11, 12, 13
application 18

Character positions 1-60 of the first line of this file contain the character set used. Their contents are intended as follows:

position	repr	symbol name
1-26	a..z	letters a through z
27-36	0..9	digits zero through nine
37		space
38	"	quote
39	#	style ii comment
40	$	formatter
41	%	percent
42	'	apostrophe
43	(	open
44	)	close
45	*	asterisk
46	+	plus
47	,	comma
48	-	minus
49	.	point
50	/	divided by
51	:	colon

52	;	semicolon
53	<	is less than
54	=	equals
55	>	is greater than
56	@	at
57	[	brief sub
58	]	brief bus
59	_	underscore
60	¬	stick

Apostrophes occur only in strings and comments.

Character positions 1-26 of the second line of this file contain the capital letters A through Z, used in strings and comments only.

Character positions 1-2 of the third line of this file contain other characters. Their contents are intended as follows:

position	repr	symbol name
1	!	exclamation mark
2	?	question mark

These characters are used only in strings and comments.

This file consists of a number of parts. The first part is this information, the following parts are the test programs. Each program is preceded by a line containing an eight-character identification and followed by a line containing 6 style-ii-comment-symbols (#). Neither of these lines is considered part of the program. No line in this file is longer than 72 characters.

The eight-character identification has the format #aaaadd# where aaaa is an abbreviation of the category of the test program and dd is the sequence number. The following abbreviations are used:

info	this information	1
null	null programs	9
simp	simple tests	16
decl	declarers	7
mdeq	mode equivalence	6
oper	operators	16
idef	identification	12
clau	clauses	9
coer	coercions	14
idrl	identity relations	2
stow	stowed values	8
flex	flexibility/transiency	4
garb	generators/garbage collection	4

scop	scope checking	10
jump	jumps	5
parl	parallel processing	6
smio	simple I/O	12
stan	standard environment	3
synt	syntax errors	8
misc	miscellaneous misery	7
numr	numerical	13
appl	application programs	19

The right-most column contains the number of programs in each category.
######

13. THE TEXT OF THE PROGRAMS

```
#null01#
BEGIN SKIP END
```

- - . - -

```
#null02#
(0)
```

- - . - -

```
#null03#
DO stop OD
```

- - . - -

```
#null04#
(SKIP, SKIP)
```

- - . - -

```
#null05#
PAR (SKIP, SKIP)
```

- - . - -

```
#null06#
BY 0 DO stop OD
```

- - . - -

```
#null07#
IF BOOL(SKIP) THEN SKIP FI
```

- - . - -

```
#null08#
CASE INT(SKIP) IN SKIP, SKIP ESAC
```

- - . - -


```
#null09#
CASE UNION(BOOL, VOID)(SKIP) IN (VOID): SKIP ESAC
```

- - . - -

```
#simp01#
BEGIN   #ALGOL68 test to see if the compiler exists#
          INT i,j,k;
          INT s=17; INT t:=3;
          i:= 0;
          FOR 1 FROM 0 BY 2 TO 13 DO i+:= 1 OD;
          print(("Value should be 42",i,newline))
END
```

- - . - -

```
#simp02#
range_of_variables:
BEGIN  #Test that ranges are correct#
          print((newline, "Values are 2,5,3,4", newline));
          INT i,j;
          i:= 3; j:= 4;
          BEGIN INT i,k;
                   i:= 2; k:= 5; print((i,k))
          END;
          print((i,j))
END
```

- - . - -

```
#simp03#
BEGIN   #Referencing and dereferencing#
          INT i1=1;
          INT i2= 2;
          INT i3= 3;
                INT ii1:= i1;
                INT ii2:= i2;
                      REF INT iii1:= ii1;
          print((newline, "Value should be 1", iii1));
          iii1:= ii2;
          print((newline, "Value should be 2", iii1))
END
```

- - . - -

```
#simp04#
multiples:
structures:
BEGIN

        #Multiple values#
        print(("Multiple values",newline));
        [1:100] INT i, j, k;
        FOR l TO 100 DO i[l]:= j[l]:= k[l]:=l OD;
        FOR l TO 100 DO
        IF i[l] /= l OR j[l] /= l OR  k[l] /= l THEN
          print("Bad multiple assignation") FI OD;
        [l : 100] REAL p;
        p[l]:= 1.0;
        p[1:5]:= (2.0, 3.0, 4.0, 5.0, 6.0);
        print ((newline, "Values are 2.0 - 6.0", newline, p[1:5]));
        print(newline);

        #Test the @ workings#
        p[2:6 #implicit @l#]:= (2.0, 3.0, 4.0, 5.0, 6.0);
        print ((newline, "Values are 2.0, 2.0 - 6.0",
                newline, p[1:6], newline));
        p[2:3@8]:=p[3:4@8];
        print((newline, "Values are 2.0, 3.0, 4.0, 4.0",
               newline, p[1:4@7], newline));
        print((newline, "Values are 11, 4",
                        UPB p[1:3@9], UPB p[1:0@5], newline));
        [1:10, 1:10] INT l;
        FOR i TO 10 DO FOR j TO 10 DO l[i,j]:=100
                                                OD OD ;
        FOR i TO  2 DO FOR j TO 10 DO l[1:2,1:10][i,j]:=
                                               11 OD OD ;
        print ((newline, "Values are 20 instances of 11 followed by ",
                         "80 of 100", newline, 1, newline));

        #Structures#
        STRUCT ([1:2] INT m,
                  [1:i[5] # whose value is 5 from above#] REAL g,
                  BOOL t) sl, s2;
        t OF sl:= t OF s2:= l[1,1] = l[1:1,1:2][1,1];  #true#
        FOR m TO UPB m OF sl DO
        (m OF sl) [m]:= (( m OF s2) [3-m]:= 50) + 1 OD;
        g OF sl:= (g OF s2)[]:= (1.0, 2.0, 3.0, 4.0, 5.0);
        print ((newline, "Structures:",
            newline, "Values are 51, 51, 1.0 to 5.0, TRUE:",
            newline, sl, newline,
            newline, "Values are 50, 50, 1.0 to 5.0, TRUE:",
            newline, s2, newline));

        #REF STRUCT's#
        [1:2] REF STRUCT ([] INT m, [] REAL g, BOOL t)
```



```
          ssl := (sl, s2);
        print((newline, "Values same as last two lines:",
                newline, ssl[1], newline, ssl[2], newline));
        t OF ssl[2]:= FALSE;
        print ((newline, "Values are TRUE, FALSE: ",
                t OF sl, t OF s2))
END
```

- - . - -

```
#simp05#
  # Simple jumps #
( INT j:= 0, i;
  k:i:= j;
    IF i >= 2 THEN GOTO 1 FI;
    print("0");
  m:IF i >= 1 THEN        n FI;
    print("0");
  o:GOTO p;
  1:print("1"); i:= i - 2; m;
  n:print("1"); o;
  p:print(newline);
    j:= j + 1;
    IF j <= 3 THEN k FI
  # Result:  00
             01
             10
             11 # )
```

- - . - -

```
#simp06#
bits bytes strings and other noise:
BEGIN   #Don't just stand there, do something!#

        print (("The following are some of the environment enquirie
                ,newline));
        print((
            "integer", int lengths, max int, newline,
            "real   ", real lengths, max real, small real, newline,
            "bits   ", bits lengths, bits width, newline,
            "bytes  ", bytes lengths, bytes width, newline,
            "null character  """", nullcharacter, """",
                                        ABSnull character,
            newline, newline));

        #bits#
        BITS a:= BIN 63 #i.e., 6 ones in a row#;
        BITS b:= BIN 1;
```

```
        print (#Let's add them and see what happens#
            ("Addition of two BITS quantities", newline,
             ABS a, ABS b, newline, "Answer should be: "));
        STRING s:= IF bits width > 6 THEN "64" ELSE "0" FI;
        BITS c:= BIN (ABS a + ABS b);
        print ((s, newline, "Answer is",ABS c, newline, newline));
        IF  2r1111111 = BIN  63
             THEN SKIP
             ELSE print ("Error in BIN things") FI;

        # reduced bytes test #
        #bytes are fixed-length strings#
        BYTES s1:= bytes pack("ab");
        [1: bytes width]CHAR cs; #to contain what the
                                                answer should be#
        cs[1:byteswidth]:= s1; s:= "ab";
        FOR i TO bytes width
        DO IF IF i<= UPB s THEN s[i]
                                ELSE null character FI /= cs[i]
            THEN
              print(("Bytes fault, values are: ",i, cs, STRING(s1)))
            ELSE
              print(("Character", i, " okay", newline))
            FI
        OD;
        print(newline);

        # Print all character values #
        print("All character values, in lines of 64 ");
        FOR i FROM 0 TO max abs char
        DO
            IF i MOD 64 = 0
           THEN print((newline,
                           whole(i, -4), "-", whole(i+63, -4), " "))
            FI;
            print(REPR i)
        OD

END
```

- - . - -

```
#simp07#
BEGIN # loops #
    INT i = 5;
    FOR i TO i DO print(i) OD;          # 1, 2, 3, 4, 5 #
    print(newline);
    print(newline);

    INT s = 8;
```



```
    FOR a FROM s BY 1 WHILE INT b = a - s + 1; a <= 2 * s
    DO INT q:= 0, r:= a;
        WHILE r >= b DO (q +:= 1, r -:= b) OD;
        IF a /= b * q + r OR r >= b
        THEN print("Error") FI;
        print((a, b, q, r, newline))
    OD;
    print(newline);                                 #   8  1  8  0
                                                        9  2  4  1
                                                       10  3  3  1
                                                       11  4  2  3
                                                       12  5  2  2
                                                       13  6  2  1
                                                       14  7  2  0
                                                       15  8  1  7
                                                       16  9  1  7   #

    PROC power 2 = (INT k) INT:
    (INT m:= 1; FOR i TO k DO m +:= power 2 (i - 1) OD; m);

    print(power 2 (6))                              # 64 #
END
```

- - . - -

```
#simp08#
  # Simple coercions #
(
  print(("Prediction:     results:", newline));

  PROC print ia = (STRING pred) VOID:
    print((pred, ": ", ia, newline));
  [1:3] INT ia:= (1, 2, 3);  print ia("+1+2+3        ");

# dereferencing #
  INT i = LOC REF INT:= ia[1] #twice dereferenced,
                                        at the right moment #;
  print(("+1           : ", i, newline));
  REF INT ri:= ia[2]; # no deref # REF INT(ri):= -2;
  print ia("+1-2+3       ");

# deproceduring #
  PROC pri = REF INT : ia[3]; pri:= -3 # soft deproc #;
  print ia("+1-2-3       ");
  PROC pria = REF [] INT: ia;  pria[1]:= pria[2];
  print ia("-2-2-3       ");

# uniting #
  UNION (REAL, []INT, [,] INT) unia = # some-uniting #
    UNION (REAL, [] INT) # cast #  # one-uniting # (ia) # deref#
```

```
  ia:= (3, 2, 1) # spoil ia # ;
  CASE unia IN
    ([] INT ia):
       (print(("-2-2-3        : ", ia, newline));
        print(("-2-2-3        : ", unia # why not ? #, newline));
        print ia("+3+2+1        ") # spoiled ia # )
    OUT print("Bad case of case")
  ESAC;

# widening #
  REAL x = ia[1]; COMPL z = x;
  print(("3e0,3e0i0e0   : ", x, z, newline));
  [] BOOL b = 8r52, STRING s = bytes pack("abc");
  print(("f...ftftftfabc: ", b, s, newline));

# rowing #
  [1:1,1:3] INT iaa; FOR i TO 3 DO iaa[1, i]:= 5 + i OD;
  PROC print iaa = (STRING pred) VOID:
  print((pred, ": ",
        LWBiaa, UPBiaa, 2LWBiaa, 2UPBiaa, iaa, newline));
  print iaa("+1+1+1+3+6+7+8");
  ia:= iaa[ 1, ]; print ia("+6+7+8        ");
  ia:= (1, 2, 3); iaa:= ia # rowing #;
  print iaa("+1+1+1+3+1+2+3");

# "hipping" #
  REF INT p = NIL, q = NIL;
  print(("true          : ", p :=: q, newline));
  ia:= (1, 5, 1) # no assignation #; 1: print ia("+1+2+3        ");
  ia:= (5, SKIP, 7); ia[2]:= 6; print ia("+5+6+7        ")
)
```

- - . - -

```
#simp09#
BEGIN # "In situ" permutation#

    PROC permvec=(REF [] INT vec, [] INT p) VOID:
    FOR j TO UPB p DO
            INT k:= p[j]; WHILE k > j DO k:= p[k] OD ;
        IF k= j THEN
            INT h= vec[j], INT l:= p[k];
            WHILE l NE j DO
                    vec[k]:= vec[l]; k:= l; l:= p[k]  OD;
            vec[k]:= h
        FI
     OD,

    [1:5] INT x:= (4, 5, 1, 3, 2);
```



```
    print(("Output: 1 2 3 4 5 ", newline));
    print((permvec(x,(3,5,4,1,2)); x))
END
```

- - . - -

```
#simp10#
( INT i:=1;
   PROC a=(INT j) VOID : print(i+j);
   (INT i:=2; a(10)
   )      #  11  #
)
```

- - . - -

```
#simp11#
BEGIN # Translation decimal number to Roman notation and vice versa #

    PROC roman = (INT number) STRING:
    BEGIN INT n:= number, STRING result,
        [] STRUCT (INT value, STRING r) table=
            ((1000, "M"), (900, "CM"), (500,"D"), (400,"CD"),
            (100,"C"), (90,"XC"), (50,"L"), (40,"XL"),
            (10,"X"), (9,"IX"), (5,"V"), (4,"IV"), (1,"I"));
        FOR i TO UPB table
        DO INT v= value OF table[i], STRING r= r OF table[i];
            WHILE v LE n DO (result +:= r, n -:= v) OD
        OD;
        result
    END,

    PROC value of roman= (STRING text) INT:
    IF text= "" THEN 0 ELSE

        OP ABS= (CHAR s) INT:
        CASE INT p; char in string (s,p, "IVXLCDM"); p IN
            1,5,10,50,100,500,1000
        ESAC,

        PROC char in string = (CHAR c, REF INT i, STRING s)
        BOOL:
        (FOR k TO UPB s DO(c = s[k]| i:= k; 1)OD; FALSE
        EXIT 1: TRUE);
        INT v, maxv:= 0, maxp;
        FOR p TO UPB text
        DO IF (v:= ABS text[p]) > maxv
            THEN maxp:= p; maxv:= v FI
        OD;
        maxv - value of roman (text[: maxp-1])
```

```
            + value of roman (text[maxp + 1:])

    FI;

    print(roman (1968)); # "MCMLXVIII" #
    print(value of roman ("MCMLXXIII")) #1973#
END
```

- - . - -

```
#simp12#
# Towers of Hanoi, Report 11.13. #
FOR k TO 8
DO FILE f:= stand out;

    PROC p= (INT me, de, ma) VOID :
    IF ma > 0 THEN
        p(me, 6 - me - de, ma - 1);
        putf(f, (me, de, ma));
        # move from peg 'me' to peg 'de' piece 'ma' #
        p(6 - me - de, de, ma - 1)
    FI;

    putf(f, ($1"k = "d1, n((2**k+15)%16)(2(2(4(3(d)x)x)x)x)1)$, k));

    p(1, 2, k)
OD
```

- - . - -

```
#simp13#
BEGIN # continued fraction #

  OP / = ([] REAL a, b) REAL :
   (UPB a=0|0|a[1]/(b[1]+a[2:]/b[2:])),

  [1:20] REAL x,y;

  FOR i TO 20 DO
  x[i]:=(i-1)**2; y[i]:= 2*i-1 OD;

  x[1]:=1;
  FOR i TO 20 DO
  print(4*(x[1:i]/y[1:i]))OD # approximations of pi #
END
```

- - . - -


```
#simp14#
( # Simple parallel program #
  [1:5] SEMA bar; FOR i TO 5 DO bar[i]:= LEVEL 0 OD;
  PAR ( # However you shuffle the cards between this one #
    (DOWN bar[3]; print(3); UP bar[4]),
    (DOWN bar[4]; print(4); UP bar[5]),
    (               print(0); UP bar[1]),
    (DOWN bar[2]; print(2); UP bar[3]),
    (DOWN bar[5]; print(5)            ),
    (DOWN bar[1]; print(1); UP bar[2]),
  # and this one, the result will always be: 0, 1, 2, 3, 4, 5 # S
)
```

- - . - -

```
#simp15#
  # uniqueness condition #
( (REAL x; x; INT x; x)                    # double x # ,
  (REAL x; x: INT x; x)                  # triple x, label in decl #
  (REAL x; INT x; REAL x; x)             # triple x # ,
  (REAL x; x: print(x))                  # double x # ,
  (x: x; x: x)                           # double x # ,

  (MODE X = REAL, Y = X, X = REAL; LOC X) # X # ,
  (PRIO X = 6; X 3)                        # no X # ,
  (PRIO X = 6, = = 7, X = 7; 3)            # double X #,
  (MODE X = Y; PRIO X = 1;3 )              # no Y, X wrong #,

  (INT a, a) INT: a                        # double a #
)
```

- - . - -

```
#simp16#
#   321   314159.265e-5   t    1.1  i 2.2
ongeluksgetal
aap:noot   654
1   2   3   4   5   6
10   20   30   40
  The above is input for the following program #
( # Simple unformatted transput #

  MODE TERMSTRING = STRUCT(STRING string, CHAR term);

  CHAR ch, INT i, REAL r, BOOL b, COMPL z,
    [ 1 : 13 ] CHAR rowch,
    STRUCT(TERMSTRING s, t, INT i) struct,
    [ 1 : 2 ] STRUCT(INT i, STRUCT(INT i, j) j) rowstruct;
  [1 : 2] INT a1, a2;
```

```
  make term(stand in, " :");

  read(ch);
  read(i);
  read(r);
  read(b);
  read(z);
  read(newline);
  print((ch, i, r,"     ", b,"    ", z, newline));
  read((rowch, newline));
  print((rowch, newline));
  read((struct, newline));
  print((struct, newline));
  read((rowstruct, newline));
  print((rowstruct, newline));

  FOR n FROM 4 BY -4 TO -4 DO  # 4, 0, -4 #
    print((newline, newline, whole(i, n),"     ", fixed(r, n, 2),
      "     ", float(r, n, 2, 2)))
  OD;

  FOR n TO 4 DO
    read(
      IF ODD n THEN a1[n OVER 2 + 1] ELSE a2[n OVER 2] FI
      )
  OD;

  print((newpage, a1, newline, a2, newline, "End"))
)
```

- - . - -

```
#decl01#
BEGIN # Some declarers #
  [1:10] INT i,
  [1:10] STRUCT (REF [] INT i, BOOL j) k,
  [1:10] STRUCT([1:10] INT i, BOOLj)l,
  [1:10] REF [] INT p,
   # formal, so no bounds allowed: #
  [1:10] PROC [1:10] INT q,
  STRUCT (REF [1:10] INT i, BOOLj) m,
  [1:10] REF [1:10] INT mn,
  PROC([1:10] INT) VOID pp,
  UNION([1:10] INT, BOOL)nm,
  [1:10] INT u=(1);
  MODE N = STRUCT(REAL a, b, a); # error, 'a' occurs twice #
  SKIP
END
```

- - . - -


```
#decl02#
BEGIN # Shielding, yin and yang #
  MODE
    Z  = Z,                          # wrong #
    A  = REF A,                      # wrong #
    B  = PROC B,                     # wrong #
    C  = STRUCT(C c),                # wrong #
    D  = PROC(D)INT,                 # right #
    E  = PROC(INT)E,                 # right #
    F  = [3] F,                      # wrong #
    G  = UNION(INT, G),              # wrong #
##
    AA = REF REF AA,                 # wrong #
    AB = REF PROC AB,                # wrong #
    AC = REF STRUCT(AC ac),          # right #
    AD = REF PROC(AD)INT,            # right #
    AE = REF PROC(INT)AE,            # right #
    AF = REF [ ] AF,                 # wrong #
    AG = REF UNION(INT, AG),         # wrong #
##
    BA = PROC REF BA,                # wrong #
    BB = PROC PROC BB,               # wrong #
    BC = PROC STRUCT(BC bc),         # right #
    BD = PROC PROC(BD)INT,           # right #
    BE = PROC PROC(INT)BE,           # right #
    BF = PROC [ ] BF,                # wrong #
    BG = PROC UNION(INT, BG),        # wrong #
##
    CA = STRUCT(REF CA ca),          # right #
    CB = STRUCT(PROC CB cb),         # right #
    CC = STRUCT(STRUCT(CC cc)cc),    # wrong #
    CD = STRUCT(PROC(CD)INT cd),     # right #
    CE = STRUCT(PROC(INT)CE ce),     # right #
    CF = STRUCT([3] CF cf),          # wrong #
    CG = STRUCT(UNION(INT, CG)cg),   # wrong #
##
    DA = PROC(REF DA)INT,            # right #
    DB = PROC(PROC DB)INT,           # right #
    DC = PROC(STRUCT(DC dc))INT,     # right #
    DD = PROC(PROC(DD)INT)INT,       # right #
    DE = PROC(PROC(INT)DE)INT,       # right #
    DF = PROC([ ] DF)INT,            # right #
    DG = PROC(UNION(INT, DG))INT,    # right #
##
    EA = PROC(INT)REF EA,            # right #
    EB = PROC(INT)PROC EB,           # right #
    EC = PROC(INT)STRUCT(EC ec),     # right #
    ED = PROC(INT)PROC(ED)INT,       # right #
    EE = PROC(INT)PROC(INT)EE,       # right #
    EF = PROC(INT)[ ] EF,            # right #
    EG = PROC(INT)UNION(INT, EG),    # right #
```

```
##
    FA = [3] REF FA,                    # wrong #
    FB = [3] PROC FB,                   # wrong #
    FC = [3] STRUCT(FC fc),             # wrong #
    FD = [3] PROC(FD)INT,               # right #
    FE = [3] PROC(INT)FE,               # right #
    FF = [3] [2] FF,                    # wrong #
    FG = [3] UNION(INT, FG),            # wrong #
##
    GA = UNION(INT, REF GA),            # wrong #
    GB = UNION(INT, PROC GB),           # wrong #
    GC = UNION(INT, STRUCT(GC gc)),     # wrong #
    GD = UNION(INT, PROC(GD)INT),       # right #
    GE = UNION(INT, PROC(INT)GE),       # right #
    GF = UNION(INT, [ ] GF),            # wrong #
    GG = UNION(INT, UNION(INT, GG));    # wrong #
    SKIP
END
```

- - . - -

```
#decl03#
BEGIN # P.G. Hibbard, Proc. Int. Conf. A68 III, Winnipeg, June, 1974:
          applied occurrence of mode-indication in actual-bounds
          of its actual-declarer #

    INT n:= 4;
    CHAR a := "a", b:= "b", c:= "c", d;

    PROC swap = (REF CHAR c1, c2) VOID:
        (d:= c1; c1:= c2; c2:= d);

    MODE HANOI =
       [IF n > 0
        THEN
            n -:= 1; swap(b, c); HANOI h1; swap(b, c);
            print((newline, "move ", whole(LWB h1 + 1, -1),
                                    " from ", a, " to ", c, "."));
            swap(a, b); HANOI h2; swap(a, b); n +:= 1
        ELSE
            0
        FI    :    1] INT;

    LOC HANOI
END
```

- - . - -


```
#decl04#
(MODE U = UNION(INT, REAL);
      LOC UNION(U) # no list needed # u := 1;
      print(u)
)
```

- - . - -

```
#decl05#
BEGIN # Application of a virtually useless mode #

  print(("Result should be: ppp", newline, newline));

  MODE P = PROC (P) P;

  P p = (P p)P: (print("p"); p);
  (p(p(p(p))))
END
```

- - . - -

```
#decl06#
BEGIN # No errors #

# 1. Actual declaration. #

# 1.a. Without bounds. #

# primitive #

BEGIN
    INT x1,
    INT x2, x3,
    INT x4, x5, x6,
    REAL x7, x8, x9, x10, x11, x12,
    LONG INT x13,
    LONG REAL x14,
    BOOL x15,
    CHAR x16;
    SKIP
END;

# ref + primitive #

BEGIN
    REF INT x1,
    REF REF REAL x2,
    REF REF REF REF LONG REAL x3,
    REF REF REF REF REF BOOL x4,
```

```
    REF REF REF REF REF REF CHAR x5,
    REF REF PROC VOID x6;
    SKIP
END;

# proc #

BEGIN
    PROC VOID x1,
    PROC PROC REAL x2,
    PROC PROC PROC PROC LONG REAL x3,
    PROC PROC PROC PROC PROC BOOL x4,
    PROC PROC PROC PROC PROC PROC CHAR x5,
    PROC PROC PROC VOID x6;
    SKIP
END;

# ref + proc #

BEGIN
    REF REF REF REF REF REF PROC VOID x1,
    PROC REF REAL x2,
    PROC REF PROC REF LONG REAL x3,
    PROC REF PROC REF PROC BOOL x4,
    REF PROC REF PROC REF PROC CHAR x5,
    PROC REF PROC VOID x6;
    SKIP
END;

# proc with one parameter which is primitive or ref + primitive #

BEGIN
    PROC(INT)VOID x1,
    PROC(REF REF REF LONG INT)VOID x2,
    PROC(BOOL)VOID x3,
    PROC(REF CHAR)VOID x4,
    PROC(LONG REAL)VOID x5,
    PROC(REF REF REF BOOL)VOID x6,
    PROC(PROC VOID)VOID x7,
    PROC(REF REF PROC VOID)VOID x8;
    SKIP
END;

# ref + proc with one parameter which is ref * primitive #

BEGIN
    REF PROC(INT)VOID x1,
    REF REF PROC(LONG REAL)VOID x2, x3,
    REF REF REF REF PROC(REF CHAR)VOID x4,
    REF PROC(PROC VOID)VOID x5,
    REF REF PROC(REF PROC VOID)VOID x6;
```



```
    SKIP
END;

# ref * proc with more than one parameter which are ref * primitive #

BEGIN
    PROC(INT, LONG INT)VOID x1,
    PROC(REAL, REF LONG REAL, REF REF BOOL)VOID x2,
    PROC(REF REF REF CHAR, INT, LONG INT, REAL, REAL,
    INT)VOID x3,
    REF PROC(INT, INT, INT, REF CHAR)VOID x4,
    REF REF REF PROC(PROC VOID, REF REF PROC VOID, INT)
        VOID x5;
    SKIP
END;

# ref + row of * ref * primitive #

BEGIN
    REF []INT x1,
    REF[,]REAL x2,
    REF REF[,,,] LONG REAL x3,
    REF REF REF[,,,,,]REF BOOL x4,
    REF[]REF REF REF LONG INT x5,
    REF REF[,,,,]REF REF CHAR x6,
    REF[]PROC VOID x7,
    REF[,]REF REF PROC VOID x8;
    SKIP
END;

# ref + row of * ref * proc #

BEGIN
    REF[] PROC VOID x1,
    REF REF[,] PROC(REF INT)VOID x2,
    REF REF REF[]PROC(INT, REF REF INT)VOID x3,
    REF[]REF PROC(REF LONG REAL, REF REF REF CHAR,
    REF LONG LONG REAL)VOID x4,
    REF REF[,,,]REF REF REF PROC(REF INT, LONG LONG
    LONG INT)VOID x5;
    REF[]PROC(PROC VOID, REF PROC VOID)VOID x6;
    SKIP
END;

# ref * proc with row of parameters #

BEGIN
    PROC([]REAL)VOID x1,
    REF PROC(INT, []LONG REAL)VOID x2,
    REF REF PROC([,]INT, []REF REF BOOL)VOID x3,
    REF REF REF REF PROC(REF[]INT, REF REF[]REF
```

```
    REF LONG REAL)VOID x4,
    PROC(REF REF[,,,]REF REF REF REAL)VOID x5,
    PROC([]REF REAL, [,,,]REF CHAR, REF[,]BOOL)VOID x6,
    PROC([]REF PROC VOID, REF[,,]PROC VOID,
        REF REF[,]REF REF PROC VOID)VOID x7;
    SKIP
END;

# nested rows #

BEGIN
    REF[,,,]REF[,]INT x1,
    REF[]REF[]PROC VOID x2,
    REF[]REF REF[]REF BOOL x3,
    REF[,,,]REF[,,,]REF REF REF PROC VOID x4,
    REF REF[]REF[]INT x5,
    REF[]REF[]REF[]REF[,]REF[]LONG REAL x6;
    SKIP
END;

# nested procs #

BEGIN
    PROC(PROC(PROC(PROC VOID)VOID)VOID)VOID x1,
    PROC(INT,
        PROC(REF REF PROC VOID,
            REAL,
            REF PROC(REF LONG INT, PROC VOID) VOID,
            INT)
          VOID)
      VOID x2,
    PROC(INT,
        PROC VOID,
        PROC(INT,
            PROC VOID,
            REF REAL,
            REF PROC VOID)
          VOID)
      VOID x3,
    PROC(INT,
        PROC(INT, INT) VOID,
        PROC(INT) VOID,
        REAL)
      VOID x4;
    SKIP
END;

# mixed rows and procs #

BEGIN
    REF[]PROC([,]INT,
```



```
        PROC([] PROC VOID) VOID,
        [,] PROC(PROC VOID, INT) VOID,
        REF[] PROC VOID)
      VOID x1;
    SKIP
END;

# 1.b. With bounds. #

BEGIN
    [1:1] INT x1,
    [1:1, 1:1] REF LONG REAL x2,
    [1:1, 1:1, 1:1]REF REF PROC VOID x3,
    [1:1]PROC VOID x4,
    [1:1]PROC(INT)VOID x5,
    [1:1]PROC(INT, REAL, REF PROC VOID)VOID x6,
    [1:1, 1:1, 1:1, 1:1]REF REF PROC(INT)VOID x7,
    [1:1]REF[]INT x8,
    [1:1]REF[,,,]REF PROC VOID x9,
    [1:1]REF PROC([]INT)VOID x10,
    [1:1]PROC([]REF PROC VOID, REF[,,]PROC VOID,
        REF REF REF REF[,]REF REF REF PROC VOID)VOID x11,
    [1:1]REF[]REF[,,]REF REF[]LONG REAL x12,
    [1:1]PROC(REF[]PROC VOID,
                    []PROC VOID, []REF INT)VOID x13;
    SKIP
END;

# 2. Variable and constant declarations. #

BEGIN
        PROC VOID a; PROC(INT)VOID b;
        PROC VOID c, d; REAL e; REAL f, g;

        SKIP; SKIP; SKIP; SKIP

END;

# 3. Declarations of routines. #

BEGIN
    INT i;
    PROC a = VOID: SKIP;
    PROC(INT)VOID b =
        (INT c)VOID: SKIP;
    PROC(INT, REAL)VOID c =
        (INT e, REAL f)VOID: SKIP;
    BEGIN PROC c = VOID: a; # no error # SKIP
    END

END;
```

```
# 4. Call without parameters. #

BEGIN
    PROC VOID a = VOID : SKIP ;
    PROC VOID b ;
    REF PROC VOID c = b ;
    REF PROC VOID d ;

    a ; # without deref #
    b ; # with deref #
    c ; # with deref #
    d ; # with deref #

    SKIP
END;

# 5. Call with parameters. #

BEGIN INT int; REAL real;
    PROC(INT)VOID dcs1,
    PROC(INT, REAL)VOID dcs2;
    PROC(INT)VOID a = (INT a)VOID: SKIP;
    PROC(INT)VOID b;
    REF PROC(INT)VOID c = b;
    REF PROC(INT)VOID d;
    PROC(INT, REAL)VOID e =
        (INT a, REAL b)VOID: SKIP;

    a(int);
    dcs1(int);
    dcs2(int, real);
    e(int, real);
    b(int);
    c(int);
    d(int);
    BEGIN REF REF REF REF REF REF REF REF REF REF
        REF REF REF REF REF REF REF REF REF REF
        PROC(INT)VOID a; a(int); SKIP END;
    BEGIN # No error #
        PROC(REF INT, REF REAL)VOID a;
        a(int, real)END;

    BEGIN PROC(INT)VOID a; a(int)END;

    BEGIN PROC(INT)VOID a; a(int); SKIP END

END;

# 6. Assignation with an identifier as destination.  #

BEGIN
```



```
        INT a; REF INT b = a; PROC(REAL)REAL c;

        a := 1;
        b := a;
        c := sin;
        SKIP

END;

# 7. Assignation with a slice as destination. #

BEGIN INT i, j, k, l;
        [ i : i ] REAL a1;
        [ i : i, j : j ] REAL a2;
        REF [] REAL a3 = a1;
        REF [,] REAL a4;
        REF [,,] REAL a5 = a4;
        [,,,] REF REAL a6 = a3[i];

        a1[i] := 3;
        a2[i, j] := 3;
        a3[i] := 3;
        a4[i, j] := 3;
        a5[i, j, k] := 3;
        a6[i, j, k, l] := 3;
    BEGIN REF REF REF REF REF REF REF REF REF REF
        REF REF REF REF REF REF REF REF REF REF
        []REAL x; x[i] :=3.0 END

END
END

                                        - - . - -


#decl07#
# Now, errors #
BEGIN

    INT int, REAL real;

        BEGIN INT a; [ a : a ] REAL b;
            [ ] REAL c; #error #
            SKIP
        END;

        BEGIN INT a, # error #
            SKIP
        END;

        BEGIN REAL x = i;  # error #
```

```
    SKIP
END;

BEGIN REAL x; REAL x;  # error #
    SKIP
END;

BEGIN REAL x = e; REAL x = e;  # error #
    SKIP
END;

BEGIN PROC a = (b #error#) VOID: SKIP ;
    SKIP
END;

BEGIN PROC a =
    (REF b #error#) VOID: SKIP;
    SKIP
END;

BEGIN PROC a = ([ i : i ] REAL x #error#) VOID: SKIP;
    SKIP
END;

BEGIN PROC (INT) VOID a = VOID: SKIP #error#;
    SKIP
END;

BEGIN PROC a = (SKIP)  #error#;
    SKIP
END;

BEGIN PROC (INT) VOID a = (INT b) SKIP #error# ;
    SKIP
END;

BEGIN REAL x; PROC x = VOID: SKIP  #error#;
    SKIP
END;

BEGIN []REAL a = int; a(int) END;

BEGIN PROC (INT) VOID a; a(a) END;

BEGIN REF [] REAL a; a (int) END;


BEGIN REF BOOL a; a(int) END;

BEGIN PROC VOID a; a(int) END;
```



```
BEGIN PROC (INT) VOID a; a(int, real) END;

BEGIN PROC (INT, REAL) VOID a; a(int) END;


    BEGIN INT x = a; x := a; SKIP; SKIP END;

    BEGIN []INT x = a; x := a END;

    BEGIN [] REF INT x = a; x := a; SKIP END;

    a := 2
# Now, errors concerning mode of primary #
BEGIN []REAL x=i; x[i]:= 3.0 END;

BEGIN REAL x; x[i]:=3.0 END;

BEGIN REF REAL x; x[i]:=3.0 END;

BEGIN REF REAL x=i; x[i]:=3.0 END;

# Now, errors concerning number of indexers #
BEGIN REF[]REAL x; x[i, j]:=3.0 END;

BEGIN REF[ , ]REAL x; x[i]:=3.0 END;

BEGIN REF[ , ]REAL x; x[i, j, k] :=3.0 END
END
```

- - . - -

```
#mdeq01#
BEGIN # Mode equivalencing #
  MODE N = UNION(STRUCT(REAL re, im), COMPL);
  # Error, modes are the same #
  SKIP
END
```

- - . - -

```
#mdeq02#
BEGIN # Mode equivalencing #

  MODE N = PROC(M)M,
         M = PROC(N)N;
  PROC M(PROC N(SKIP)); PROC N(PROC M(SKIP));
  # Both okay, since 'M' and 'N' are the same #

  SKIP
```

```
END
```

- - . - -

```
#mdeq03#
BEGIN # Mode equivalencing #
  MODE M = PROC(M)M,
         N = PROC(N)N,
         O = UNION(N,M); # error, 'M' and 'N' are the same #
  SKIP
END
```

- - . - -

```
#mdeq04#
BEGIN # Unions #

  MODE N = UNION (REAL, UNION (BOOL, INT)),
         M = UNION (UNION(REAL, BOOL), INT);
  PROC M(PROC N(SKIP)); PROC N(PROC M(SKIP));
  # both okay, since 'M' and 'N' are the same #

  MODE U = UNION (INT, PROC(U) INT),
         V = UNION(U, PROC(V) INT);
  PROC U(PROC V(SKIP)); PROC V(PROC U(SKIP));
  # both okay, since 'U' and 'V' are the same #

  SKIP
END
```

- - . - -

```
#mdeq05#
BEGIN # Mode equivalencing #
  MODE N = UNION(BYTES, BITS, REF BITS);
                # error, related #
   MODE SZEREDI = UNION(INT,REAL,REF UNION(INT,REAL))
   #  Szeredi - ambiguity  #;
   SKIP
END
```

- - . - -

```
#mdeq06#
BEGIN # Some equivalencing #

  MODE A = STRUCT(REF A l, REF A r),
```



```
         B = STRUCT(REF B l, REF B r),
         C = STRUCT(REF D l, REF E r),
         D = STRUCT(REF E l, REF C r),
         E = STRUCT(REF C l, REF D r),
         F = STRUCT(REF STRUCT(REF A l, REF B r) l,
                    REF STRUCT
                        (REF STRUCT(REF C l, REF D r) l,
                         REF STRUCT(REF E l, REF F r) r
                        ) r
                   );

  MODE M = UNION(A, B, C, D, E, F);
  # error, all modes are the same #

  SKIP
END
```

- - . - -

```
#oper01#
BEGIN # Operator test #

  OP +:= = (INT a,b) INT : a+b;
  OP +:= = (INT a, REAL b) INT :ROUND(a-b);

  print(2+:=1); # yields 3 #
  print(2+:=1.0)  # yields 1 #
END
```

- - . - -

```
#oper02#
BEGIN # Operator #
  OP + =(UNION(INT, BOOL)a)INT: (a|(BOOL):1, (INT):2);
  print(+IF TRUE THEN TRUE ELSE 0 FI); # 1 #
  print(+IF FALSE THEN TRUE ELSE 0 FI)  # 2 #
END
```

- - . - -

```
#oper03#
BEGIN # Priorities #
  PRIO + = 7; print(1+2*3); # 9 #
  BEGIN PRIO + = 6; print(1 +2*3) # 7 #;
    FOR i TO 1 WHILE PRIO + = 7; TRUE DO
    print(1+2*3) # 9 # OD;
    print(1+2*3) # 7 #
  END;
```

```
  print(1+2*3) # 9 #
END
```

- - . - -

```
#oper04#
BEGIN # Operator identification #

  MODE M = UNION ([]INT, BOOL, STRING);

  OP +=(REALa)INT:2,
  OP +=(CHARa)INT:3,
  OP +=(M   a)INT:1;

  PROC prpm = REF PROC M: HEAP PROC M:= M : "aap";
  UNION (BOOL, STRING) b =  "b ";

  FOR n TO 5 DO
print(+  CASE n IN SKIP, TRUE, IF FALSE THEN "aa" ELSE
    b FI, prpm OUT LOC[1:1]INT:=1 ESAC)OD
  # yields 11111 #
END
```

- - . - -

```
#oper05#
BEGIN # Operator test, mutual recursion #
  PRIO +>=1,+<=1;
  OP +>= (INT a, b) INT:a+<b;
  OP +<= (INT a, b) INT:a+>b;
  1+>2 # loop #
END
```

- - . - -

```
#oper06#
BEGIN # operator #
  OP +=(REAL a, b)REAL: a-b,
  OP +=(REF REAL a,b)REAL: a-b; # error, related modes #
  SKIP
END
```

- - . - -

```
#oper07#
( # Operator declarations #
  OP SQ = (REAL x) REAL: x * x,
```



```
      RD = (INT i) REAL : random,
  OP (REAL) REAL SIN = ( print("Print ten times 1"); sin),
                         COS = cos;

  print(newline);
  TO 10 DO
   print(BEGIN REAL x=RD 1;SQ SIN x +SQ COS x END)OD
)
```

- - . - -

```
#oper08#
( # A complicated formula relying totally on priorities #

  OP I= (INT i, j) COMPL : (i, j);
  OP ** = (INT i, COMPL z) INT : ROUND(i + RE z + IM z);
  OP <  = (INT i, j) INT : (i - j) * 2;
  OP =  = (INT i, j) INT : (i + j) * 2;
  OP AND= (INT i, j) INT : (i + i - j) * 3;
  OP OR= (INT i, j) INT : (i - j - j) * 3;

  INT loc int;
  # Note: all operators are followed by their priorities #
  print((loc int := 0) -:= 1 OR 2 AND 3 = 4 < 5 + 6 * 7 ** 8 I 9
    -:= 1 ** 8 OR 2 * 7 AND 3 + 6 = 4 < 5)

  # The implied parenthesis structure is :
    (1(2(3(4(5(6(7(8(9)))))))))1((8)2((7)3((6)4(5))))
    and it yields 10650 #
)
```

- - . - -

```
#oper09#
BEGIN # Monadic operators, non-bold monads # INT decls := 0;

    OP    += (INT a) INT: a + 1; decls +:= 1;
    OP   +<= (INT a) INT: a + 1; decls +:= 1;
    OP   +>= (INT a) INT: a + 1; decls +:= 1;
    OP   +/= (INT a) INT: a + 1; decls +:= 1;
    OP   +== (INT a) INT: a + 1; decls +:= 1;
    OP   +*= (INT a) INT: a + 1; decls +:= 1;
    OP  +:== (INT a) INT: a + 1; decls +:= 1;
    OP +<:== (INT a) INT: a + 1; decls +:= 1;
    OP +>:== (INT a) INT: a + 1; decls +:= 1;
    OP +/:== (INT a) INT: a + 1; decls +:= 1;
    OP +=:== (INT a) INT: a + 1; decls +:= 1;
    OP +*:== (INT a) INT: a + 1; decls +:= 1;
    OP  +=:= (INT a) INT: a + 1; decls +:= 1;
```

```
OP +<=:= (INT a) INT: a + 1; decls +:= 1;
OP +>=:= (INT a) INT: a + 1; decls +:= 1;
OP +/=:= (INT a) INT: a + 1; decls +:= 1;
OP +==:= (INT a) INT: a + 1; decls +:= 1;
OP +*=:= (INT a) INT: a + 1; decls +:= 1;

OP    -= (INT a) INT: a + 1; decls +:= 1;
OP   -<= (INT a) INT: a + 1; decls +:= 1;
OP   ->= (INT a) INT: a + 1; decls +:= 1;
OP   -/= (INT a) INT: a + 1; decls +:= 1;
OP   -== (INT a) INT: a + 1; decls +:= 1;
OP   -*= (INT a) INT: a + 1; decls +:= 1;
OP  -:== (INT a) INT: a + 1; decls +:= 1;
OP -<:== (INT a) INT: a + 1; decls +:= 1;
OP ->:== (INT a) INT: a + 1; decls +:= 1;
OP -/:== (INT a) INT: a + 1; decls +:= 1;
OP -=:== (INT a) INT: a + 1; decls +:= 1;
OP -*:== (INT a) INT: a + 1; decls +:= 1;
OP  -=:= (INT a) INT: a + 1; decls +:= 1;
OP -<=:= (INT a) INT: a + 1; decls +:= 1;
OP ->=:= (INT a) INT: a + 1; decls +:= 1;
OP -/=:= (INT a) INT: a + 1; decls +:= 1;
OP -==:= (INT a) INT: a + 1; decls +:= 1;
OP -*=:= (INT a) INT: a + 1; decls +:= 1;

OP    %= (INT a) INT: a + 1; decls +:= 1;
OP   %<= (INT a) INT: a + 1; decls +:= 1;
OP   %>= (INT a) INT: a + 1; decls +:= 1;
OP   %/= (INT a) INT: a + 1; decls +:= 1;
OP   %== (INT a) INT: a + 1; decls +:= 1;
OP   %*= (INT a) INT: a + 1; decls +:= 1;
OP  %:== (INT a) INT: a + 1; decls +:= 1;
OP %<:== (INT a) INT: a + 1; decls +:= 1;
OP %>:== (INT a) INT: a + 1; decls +:= 1;
OP %/:== (INT a) INT: a + 1; decls +:= 1;
OP %=:== (INT a) INT: a + 1; decls +:= 1;
OP %*:== (INT a) INT: a + 1; decls +:= 1;
OP  %=:= (INT a) INT: a + 1; decls +:= 1;
OP %<=:= (INT a) INT: a + 1; decls +:= 1;
OP %>=:= (INT a) INT: a + 1; decls +:= 1;
OP %/=:= (INT a) INT: a + 1; decls +:= 1;
OP %==:= (INT a) INT: a + 1; decls +:= 1;
OP %*=:= (INT a) INT: a + 1; decls +:= 1;

print((
"Should print two equal integers (number of non-bold monads)",
newline,
++<+>+/+=+*+:=+<:=+>:=+,:=+=:=+*:=+=:+<=:+>=:+/=:+==:+*=:
--<->-/-=-*-:=-<:=->:=-/:=-=:=-*:=-=:-<=:->=:-/=:-==:-*=:
%%<%>%/%=%*%:=%<:=%>:=%/:=%=:=%*:=%=:%<=:%>=:%/=:%==:%*=:
0, decls))
```



```
END
```

- - . - -

```
#oper10#
BEGIN # Dyadic operators, non-bold monads # INT decls := 0;

    # the first declaration is different to avoid a recursive loop #
    OP     += (INT a, b) INT: (INT c:= a; c PLUSAB b);
         decls PLUSAB 1;
    OP    +<= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP    +>= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP    +/= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP    +== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP    +*= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP   +:== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  +<:== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  +>:== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  +/:== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  +=:== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  +*:== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP   +=:= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  +<=:= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  +>=:= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  +/=:= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  +==:= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  +*=:= (INT a, b) INT: a + b; decls PLUSAB 1;

    OP     -= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP    -<= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP    ->= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP    -/= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP    -== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP    -*= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP   -:== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  -<:== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  ->:== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  -/:== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  -=:== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  -*:== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP   -=:= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  -<=:= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  ->=:= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  -/=:= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  -==:= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  -*=:= (INT a, b) INT: a + b; decls PLUSAB 1;

    OP     %= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP    %<= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP    %>= (INT a, b) INT: a + b; decls PLUSAB 1;
```

```
    OP    %/= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP    %== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP    %*= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP   %:== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  %<:== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  %>:== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  %/:== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  %=:== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  %*:== (INT a, b) INT: a + b; decls PLUSAB 1;
    OP   %=:= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  %<=:= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  %>=:= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  %/=:= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  %==:= (INT a, b) INT: a + b; decls PLUSAB 1;
    OP  %*=:= (INT a, b) INT: a + b; decls PLUSAB 1;

    PRIO
    +=1, +<=1, +>=1, +/=1, +==1, +*=1,
    +:==1, +<:==1, +>:==1, +/:==1, +=:==1, +*:==1,
    +=:=1, +<=:=1, +>=:=1, +/=:=1, +==:=1, +*=:=1,

    -=1, -<=1, ->=1, -/=1, -==1, -*=1,
    -:==1, -<:==1, ->:==1, -/:==1, -=:==1, -*:==1,
    -=:=1, -<=:=1, ->=:=1, -/=:=1, -==:=1, -*=:=1,

    %=1, %<=1, %>=1, %/=1, %==1, %*=1,
    %:==1, %<:==1, %>:==1, %/:==1, %=:==1, %*:==1,
    %=:=1, %<=:=1, %>=:=1, %/=:=1, %==:=1, %*=:=1;

print((
    "Should print two equal integers (number of non-bold monads)",
    newline, 0
    + 1  +< 1  +> 1  +/ 1  += 1  +* 1
    +:= 1  +<:= 1  +>:= 1  +/:= 1  +=:= 1  +*:= 1
    +=: 1  +<=: 1  +>=: 1  +/=: 1  +==: 1  +*=: 1

    - 1  -< 1  -> 1  -/ 1  -= 1  -* 1
    -:= 1  -<:= 1  ->:= 1  -/:= 1  -=:= 1  -*:= 1
    -=: 1  -<=: 1  ->=: 1  -/=: 1  -==: 1  -*=: 1

    % 1  %< 1  %> 1  %/ 1  %= 1  %* 1
    %:= 1  %<:= 1  %>:= 1  %/:= 1  %=:= 1  %*:= 1
    %=: 1  %<=: 1  %>=: 1  %/=: 1  %==: 1  %*=: 1
    , decls))
END
```

- - . - -


```
#oper11#
BEGIN # Dyadic operators, non-bold nomads # INT decls := 0;

    OP    <= (INT a, b) INT: a + b; decls +:= 1;
    OP   <<= (INT a, b) INT: a + b; decls +:= 1;
    OP   <>= (INT a, b) INT: a + b; decls +:= 1;
    OP   </= (INT a, b) INT: a + b; decls +:= 1;
    OP   <== (INT a, b) INT: a + b; decls +:= 1;
    OP   <*= (INT a, b) INT: a + b; decls +:= 1;
    OP  <:== (INT a, b) INT: a + b; decls +:= 1;
    OP <<:== (INT a, b) INT: a + b; decls +:= 1;
    OP <>:== (INT a, b) INT: a + b; decls +:= 1;
    OP </:== (INT a, b) INT: a + b; decls +:= 1;
    OP <=:== (INT a, b) INT: a + b; decls +:= 1;
    OP <*:== (INT a, b) INT: a + b; decls +:= 1;
    OP  <=:= (INT a, b) INT: a + b; decls +:= 1;
    OP <<=:= (INT a, b) INT: a + b; decls +:= 1;
    OP <>=:= (INT a, b) INT: a + b; decls +:= 1;
    OP </=:= (INT a, b) INT: a + b; decls +:= 1;
    OP <==:= (INT a, b) INT: a + b; decls +:= 1;
    OP <*=:= (INT a, b) INT: a + b; decls +:= 1;

    OP    >= (INT a, b) INT: a + b; decls +:= 1;
    OP   ><= (INT a, b) INT: a + b; decls +:= 1;
    OP   >>= (INT a, b) INT: a + b; decls +:= 1;
    OP   >/= (INT a, b) INT: a + b; decls +:= 1;
    OP   >== (INT a, b) INT: a + b; decls +:= 1;
    OP   >*= (INT a, b) INT: a + b; decls +:= 1;
    OP  >:== (INT a, b) INT: a + b; decls +:= 1;
    OP ><:== (INT a, b) INT: a + b; decls +:= 1;
    OP >>:== (INT a, b) INT: a + b; decls +:= 1;
    OP >/:== (INT a, b) INT: a + b; decls +:= 1;
    OP >=:== (INT a, b) INT: a + b; decls +:= 1;
    OP >*:== (INT a, b) INT: a + b; decls +:= 1;
    OP  >=:= (INT a, b) INT: a + b; decls +:= 1;
    OP ><=:= (INT a, b) INT: a + b; decls +:= 1;
    OP >>=:= (INT a, b) INT: a + b; decls +:= 1;
    OP >/=:= (INT a, b) INT: a + b; decls +:= 1;
    OP >==:= (INT a, b) INT: a + b; decls +:= 1;
    OP >*=:= (INT a, b) INT: a + b; decls +:= 1;

    OP    /= (INT a, b) INT: a + b; decls +:= 1;
    OP   /<= (INT a, b) INT: a + b; decls +:= 1;
    OP   />= (INT a, b) INT: a + b; decls +:= 1;
    OP   //= (INT a, b) INT: a + b; decls +:= 1;
    OP   /== (INT a, b) INT: a + b; decls +:= 1;
    OP   /*= (INT a, b) INT: a + b; decls +:= 1;
    OP  /:== (INT a, b) INT: a + b; decls +:= 1;
    OP /<:== (INT a, b) INT: a + b; decls +:= 1;
    OP />:== (INT a, b) INT: a + b; decls +:= 1;
    OP //:== (INT a, b) INT: a + b; decls +:= 1;
```

```
OP /=:== (INT a, b) INT: a + b; decls +:= 1;
OP /*:== (INT a, b) INT: a + b; decls +:= 1;
OP  /=:= (INT a, b) INT: a + b; decls +:= 1;
OP /<=:= (INT a, b) INT: a + b; decls +:= 1;
OP />=:= (INT a, b) INT: a + b; decls +:= 1;
OP //=:= (INT a, b) INT: a + b; decls +:= 1;
OP /==:= (INT a, b) INT: a + b; decls +:= 1;
OP /*=:= (INT a, b) INT: a + b; decls +:= 1;

OP    == (INT a, b) INT: a + b; decls +:= 1;
OP   =<= (INT a, b) INT: a + b; decls +:= 1;
OP   =>= (INT a, b) INT: a + b; decls +:= 1;
OP   =/= (INT a, b) INT: a + b; decls +:= 1;
OP   === (INT a, b) INT: a + b; decls +:= 1;
OP   =*= (INT a, b) INT: a + b; decls +:= 1;
OP  =:== (INT a, b) INT: a + b; decls +:= 1;
OP =<:== (INT a, b) INT: a + b; decls +:= 1;
OP =>:== (INT a, b) INT: a + b; decls +:= 1;
OP =/:== (INT a, b) INT: a + b; decls +:= 1;
OP ==:== (INT a, b) INT: a + b; decls +:= 1;
OP =*:== (INT a, b) INT: a + b; decls +:= 1;
OP  ==:= (INT a, b) INT: a + b; decls +:= 1;
OP =<=:= (INT a, b) INT: a + b; decls +:= 1;
OP =>=:= (INT a, b) INT: a + b; decls +:= 1;
OP =/=:= (INT a, b) INT: a + b; decls +:= 1;
OP ===:= (INT a, b) INT: a + b; decls +:= 1;
OP =*=:= (INT a, b) INT: a + b; decls +:= 1;

OP    *= (INT a, b) INT: a + b; decls +:= 1;
OP   *<= (INT a, b) INT: a + b; decls +:= 1;
OP   *>= (INT a, b) INT: a + b; decls +:= 1;
OP   */= (INT a, b) INT: a + b; decls +:= 1;
OP   *== (INT a, b) INT: a + b; decls +:= 1;
OP   **= (INT a, b) INT: a + b; decls +:= 1;
OP  *:== (INT a, b) INT: a + b; decls +:= 1;
OP *<:== (INT a, b) INT: a + b; decls +:= 1;
OP *>:== (INT a, b) INT: a + b; decls +:= 1;
OP */:== (INT a, b) INT: a + b; decls +:= 1;
OP *=:== (INT a, b) INT: a + b; decls +:= 1;
OP **:== (INT a, b) INT: a + b; decls +:= 1;
OP  *=:= (INT a, b) INT: a + b; decls +:= 1;
OP *<=:= (INT a, b) INT: a + b; decls +:= 1;
OP *>=:= (INT a, b) INT: a + b; decls +:= 1;
OP */=:= (INT a, b) INT: a + b; decls +:= 1;
OP *==:= (INT a, b) INT: a + b; decls +:= 1;
OP **=:= (INT a, b) INT: a + b; decls +:= 1;

PRIO
<=1, <<=1, <>=1, </=1, <==1, <*=1,
<:==1, <<:==1, <>:==1, </:==1, <=:==1, <*:==1,
<=:=1, <<=:=1, <>=:=1, </=:=1, <==:=1, <*=:=1,
```



```
    >=1, ><=1, >>=1, >/=1, >==1, >*=1,
    >:==1, ><:==1, >>:==1, >/:==1, >=:==1, >*:==1,
    >=:=1, ><=:=1, >>=:=1, >/=:=1, >==:=1, >*=:=1,

    /=1, /<=1, />=1, //=1, /==1, /*=1,
    /:==1, /<:==1, />:==1, //:==1, /=:==1, /*:==1,
    /=:=1, /<=:=1, />=:=1, //=:=1, /==:=1, /*=:=1,

    ==1, =<=1, =>=1, =/=1, ===1, =*=1,
    =:==1, =<:==1, =>:==1, =/:==1, ==:==1, =*:==1,
    ==:=1, =<=:=1, =>=:=1, =/=:=1, ===:=1, =*=:=1,

    *=1, *<=1, *>=1, */=1, *==1, **=1,
    *:==1, *<:==1, *>:==1, */:==1, *=:==1, **:==1,
    *=:=1, *<=:=1, *>=:=1, */=:=1, *==:=1, **=:=1;

    print((
    "Should print two equal integers (number of non-bold nomads)",
    newline, 0
    < 1  << 1  <> 1  </ 1  <= 1  <* 1
    <:= 1  <<:= 1  <>:= 1  </:= 1  <=:= 1  <*:= 1
    <=: 1  <<=: 1  <>=: 1  </=: 1  <==: 1  <*=: 1

    > 1  >< 1  >> 1  >/ 1  >= 1  >* 1
    >:= 1  ><:= 1  >>:= 1  >/:= 1  >=:= 1  >*:= 1
    >=: 1  ><=: 1  >>=: 1  >/=: 1  >==: 1  >*=: 1

    / 1  /< 1  /> 1  // 1  /= 1  /* 1
    /:= 1  /<:= 1  />:= 1  //:= 1  /=:= 1  /*:= 1
    /=: 1  /<=: 1  />=: 1  //=: 1  /==: 1  /*=: 1

    = 1  =< 1  => 1  =/ 1  == 1  =* 1
    =:= 1  =<:= 1  =>:= 1  =/:= 1  ==:= 1  =*:= 1
    ==: 1  =<=: 1  =>=: 1  =/=: 1  ===: 1  =*=: 1

    * 1  *< 1  *> 1  */ 1  *= 1  ** 1
    *:= 1  *<:= 1  *>:= 1  */:= 1  *=:= 1  **:= 1
    *=: 1  *<=: 1  *>=: 1  */=: 1  *==: 1  **=: 1
    , decls))

END
```

- - . - -

```
#oper12#
BEGIN # Operator test, illegal operator #
  OP  += = (INT a) INT : -a;
  OP +:= = (INT a) INT : -a;
  OP-/:= = (INT a) INT : -a;
  OP+==: = (INT a) INT : -a;
```

```
  # Correct version: #
  print(+=+:=-/:=+==:+==:+=+:=+==:1);
  # Bad version #
  print(+=+:=-/:=+==:+==+=+:=+==:=1)
END
```

- - . - -

```
#oper13#
BEGIN # Operator test, illegal #

  OP +=<== (INT a) INT :-a;                                  # bad #

  OP <==(INTi) INT : -i;                          # <= is dyadic only #

  OP +==(INTa,b,c)INT:1;                                  # incorrect #
  OP +==:=(INTa,b,c)INT:1;                                # incorrect #
  OP ===(INTa)INT:1;                                      # incorrect #
  OP ==:=(INTa)INT:1;                                     # incorrect #

  # The ##= is intended as the "differs from symbol"
    which is not available in the Standard Hardware Representation #
  INTa,b;
  (a:=b);                                 # correct, assignation #
  (a:##=b);                               # correct, label, mon. op #
  (a:/=b);                                # incorrect #

  (a:=:b);                                # correct, IS #
  (a:##=:b);                              # correct, ISNT #
  (a:/=:b);                               # correct, ISNT #

  (a:=:=b);                               # incorrect #
  (a:##=:=b);                             # correct, label, mon. op #
  (a:/=:=b);                              # incorrect #

  OP +:= = (INT a,b) INT : a+b;
  OP +:= = (INT a, REAL b) INT :ROUND(a-b);

  UNION(INT, REAL) i:= 1;
  print(2+:= i)                  # error, operator cannot be identified #
END
```

- - . - -

```
#oper14#
BEGIN

  print(("Results must be:", newline, 4, 5, 5, 4, newline,
         1, 2, 2, newline, 1, 1, 1, newline,
```



```
         2, 1, 3, newline, 1, newline,
         1, 1, 1, 1, newline, newline,
         "Results are:", newline));

  print((UPB []INT(1, 2, 3, 4),  UPB "abcde",
         UPB []INT(SKIP, SKIP) [1 : 1 @ 5],
         2 UPB [,]INT(1, 2) [, @ 4]));

  print(newline);
  # All declarers are of the mode row-of, so UPB/LWB should work #
  print((UPB UNION ([] INT, [,] INT) ([] INT (1)),
    UPB UNION ([] INT, [,] INT) ([,] INT (1,1)),
    UPB UNION ([] STRING, STRING) ("ab")));

  print(newline);
  print((LWB UNION ([] INT, [,] INT) ([] INT (1)),
    LWB UNION ([] INT, [,] INT) ([,] INT (1,1)),
    LWB UNION ([] STRING, STRING) ("ab")));

  print(newline);
  FOR i TO 3
  DO print(i UPB [,,] CHAR ("abc", "def")) OD;

  print(newline);
  print(LWB LOC STRING LWB LOC STRING);

  print(newline);
  # Balance #
  FOR n TO 4 DO
    print(n UPB
      CASE n IN
        [] INT (1),
        [,] REAL (1),
        UNION([]INT, [,,]BOOL) ([,,]BOOL (TRUE)),
        UNION([]INT, UNION([,]REAL, [,,,]CHAR))
                                              ([,,,]CHAR ("a"))
      ESAC
    )
  OD

END
```

- - . - -

```
#oper15#
BEGIN
  # Incorrect, since not all declarers are of the mode row-of #
  print(UPB UNION ([] INT, BOOL) ([] INT (1)));
  print(LWB UNION (REF [,] STRING, STRING) ("ab"))
END
```

- - . - -

```
#oper16#
#    Tests  on  operators          #
BEGIN

    PROC error = (INT i) VOID:
    print(("Error in test", i));

    PROC tste = (INT i) VOID:
    BEGIN error(i);
        print(("; wrong branch taken", newline))
    END;

    PROC tsti = (INT i, INT p, q) VOID:
    IF p = q THEN SKIP
    ELSE error(i);
        print(("; value is: ", q, ", must be: ", p, newline))
    FI;

    PROC tstr = (INT i, REAL p, q) VOID:
    # two reals are considered equal if their difference is negligible
      compared to one of them
    #
    IF p + (p-q)/8 = p THEN SKIP
    ELSE error(i);
        print(("; value is: ", q, ", must be: ", p, newline))
    FI;

    PROC tstb = (INT i, BOOL p, q) VOID:
    BEGIN
        IF p THEN IF q THEN SKIP ELSE GOTO bad FI
               ELSE IF q THEN GOTO bad ELSE SKIP FI
        FI
    EXIT bad:
        error(i);
        print(("; value is: ", q, ", must be: ", p, newline))
    END;

    PROC tstc = (INT i, CHAR p, q) VOID:
    IF p = q THEN SKIP
    ELSE error(i);
        print(("; value is: ", q, ", must be: ", p, newline))
    FI;

    PROC tstli = (INT i, LONG INT p, q) VOID:
    IF p = q THEN SKIP
    ELSE error(i);
        print(("; value is: ", q, ", must be: ", p, newline))
    FI;
```



```
    PROC tstlr = (INT i, LONG REAL p, q) VOID:
    IF p + (p-q)/LONG 8 = p THEN SKIP
    ELSE error(i);
        print(("; value is: ", q, ", must be: ", p, newline))
    FI;

print(("Test: REPR, ABS", newline));
BEGIN
    INT b1; LONG INT b2;
    INT b0 = 44;
    [1:2]CHAR a;
    a[2]:="a";
    tstc(1, "a", REPR ABS"a");
    tstc(2, "a", REPR ABS a[2]);
    tsti(3, +43, ABS(REPR 43));
    tsti(4, +44, ABS(REPR b0));
    b1:=45;
    tsti(5, +45, ABS(REPR b1));
    tsti(6, +46, ABS(REPR (46+0)));
    b2:=LONG 43;
    tsti(7, +43, ABS(REPR SHORTEN b2));
    tsti(8, +46, ABS REPR SHORTEN LONG 46)
END;

print(("Test: LWB, UPB", newline));
BEGIN
    REF[]CHAR b;
    [-5:-3, -1:3]REF[ , ]REAL a;
    tsti(9, -5, 1 LWB a);
    tsti(10, -5, LWB a);
    tsti(11, -1, 2 LWB a);
    tsti(12, -3, 1 UPB a);
    tsti(13, -3, UPB a);
    tsti(14, +3, 2 UPB a);
    FOR i FROM LWB a BY 1 TO UPB a DO
    FOR j FROM 2 LWB a BY 1 TO 2 UPB a DO
        BEGIN
            [i:j, -j:-i]REAL b;
            a[i, j] := b;
            tsti(15,  i, 1 LWB a[i, j]);
            tsti(16,  j, UPB a[i, j]);
            tsti(17, -j, 2 LWB a[i, j]);
            tsti(18, -i, 2 UPB a[i, j])
        END
    OD OD
END;

BEGIN
    [1:3, 2:4, 3:5]INT a aaaaa;
    INT jjjjjj;
    FOR i FROM 1 BY 1 TO 3
```

```
    DO tsti(19, i, i LWB aaaaaa);
            tsti(20, i+2, i UPB aaaaaa)
    OD;
    FOR i FROM-3BY 1 TO-1
    DO tsti(21, -i, -i LWB aaaaaa);
            tsti(22, 2-i, -i UPB aaaaaa)
    OD;
    jjjjjj:=2; tsti(23, +2, jjjjjj LWB aaaaaa);
               tsti(24, +4, jjjjjj UPB aaaaaa);
    tsti(25, +1, LWB"abc");
    tsti(26, +3, UPB"cde");
    tsti(27, +1, (1+0) LWB"abc");
    tsti(28, +3, 1 UPB"efg")
END;

print(("Test: OR, AND", newline));
BEGIN
    BOOL t = TRUE; BOOL f = FALSE;
    BOOL a;
    a:=t OR t; tstb(29, TRUE, a);
    a:=t OR f; tstb(30, TRUE, a);
    a:=f OR t; tstb(31, TRUE, a);
    a:=f OR f; tstb(32, FALSE, a);
    a:=t AND t; tstb(33, TRUE, a);
    a:=t AND f; tstb(34, FALSE, a);
    a:=f AND t; tstb(35, FALSE, a);
    a:=f AND f; tstb(36, FALSE, a);
    a:=NOT t AND t; tstb(37, FALSE, a);
    a:=NOT f AND t; tstb(38, TRUE, a);
    a:=NOT t AND f; tstb(39, FALSE, a);
    a:=NOT f AND f; tstb(40, FALSE, a);
    a:=NOT t OR t; tstb(41, TRUE, a);
    a:=NOT f OR t; tstb(42, TRUE, a);
    a:=NOT t OR f; tstb(43, FALSE, a);
    a:=NOT f OR f; tstb(44, TRUE, a);
    a:=t AND NOT t; tstb(45, FALSE, a);
    a:=t AND NOT f; tstb(46, TRUE, a);
    a:=f AND NOT t; tstb(47, FALSE, a);
    a:=f AND NOT f; tstb(48, FALSE, a);
    a:=t OR NOT t; tstb(49, TRUE, a);
    a:=t OR NOT f; tstb(50, TRUE, a);
    a:=f OR NOT t; tstb(51, FALSE, a);
    a:=f OR NOT f; tstb(52, TRUE, a);
    a:=NOT t AND NOT t; tstb(53, FALSE, a);
    a:=NOT t AND NOT f; tstb(54, FALSE, a);
    a:=NOT f AND NOT t; tstb(55, FALSE, a);
    a:=NOT f AND NOT f; tstb(56, TRUE, a);
    a:=NOT t OR NOT t; tstb(57, FALSE, a);
    a:=NOT t OR NOT f; tstb(58, TRUE, a);
    a:=NOT f OR NOT t; tstb(59, TRUE, a);
    a:=NOT f OR NOT f; tstb(60, TRUE, a);
```



```
    a:=t; tstb(61, TRUE, a);
    a:=a AND t; tstb(62, TRUE, a);
    a:=a OR t; tstb(63, TRUE, a);
    a:=a OR f; tstb(64, TRUE, a);
    a:=a AND f; tstb(65, FALSE, a);
    a:=a AND f; tstb(66, FALSE, a);
    a:=a AND t; tstb(67, FALSE, a);
    a:=a OR f; tstb(68, FALSE, a);
    a:=a OR t; tstb(69, TRUE, a);
    a:=t; tstb(70, TRUE, a);
    a:=t AND a; tstb(71, TRUE, a);
    a:=t OR a; tstb(72, TRUE, a);
    a:=f OR a; tstb(73, TRUE, a);
    a:=f AND a; tstb(74, FALSE, a);
    a:=f AND a; tstb(75, FALSE, a);
    a:=t AND a; tstb(76, FALSE, a);
    a:=f OR a; tstb(77, FALSE, a);
    a:=t OR a; tstb(78, TRUE, a);
    SKIP
END;

BEGIN
    BOOL t = TRUE; BOOL f = FALSE;
    tstb(79, TRUE, t OR f);
    tstb(80, TRUE, t OR t);
    tstb(81, TRUE, f OR t);
    tstb(82, FALSE, f OR f);
    tstb(83, TRUE, NOT (f OR f));
    tstb(84, TRUE, NOT (f AND f));
    tstb(85, TRUE, t AND t);
    tstb(86, FALSE, t AND f);
    tstb(87, FALSE, f AND t);
    tstb(88, FALSE, f AND f);
    tstb(89, TRUE, (t OR t) OR (f OR f));
    tstb(90, FALSE, (t OR t) AND (f OR f));
    tstb(91, TRUE, t OR (f OR f));
    tstb(92, TRUE, NOT (t AND (f OR f)));
    tstb(93, FALSE, NOT NOT ((f OR f) OR f));
    tstb(94, TRUE, NOT NOT NOT ((f OR f) AND f));
    IF t OR f THEN SKIP ELSE tste(95) FI;
    IF t OR t THEN SKIP ELSE tste(96) FI;
    IF f OR t THEN SKIP ELSE tste(97) FI;
    IF f OR f THEN tste(98) FI;
    IF NOT (f OR f) THEN SKIP ELSE tste(99) FI;
    IF NOT (f AND f) THEN SKIP ELSE tste(100) FI;
    IF t AND t THEN SKIP ELSE tste(101) FI;
    IF t AND f THEN tste(102) FI;
    IF f AND t THEN tste(103) FI;
    IF f AND f THEN tste(104) FI;
    IF (t OR t) AND (t OR t) THEN SKIP ELSE tste(105) FI;
    IF (t OR t) OR (t OR t) THEN SKIP ELSE tste(106) FI;
```

```
    IF (t OR t) OR f THEN SKIP ELSE tste(107) FI;
    IF (t OR t) AND t THEN SKIP ELSE tste(108) FI;
    IF t OR (t OR t) THEN SKIP ELSE tste(109) FI;
    IF t AND (t OR f) THEN SKIP ELSE tste(110) FI;
    BOOL a1; a1:=t AND f; tstb(111, FALSE, a1);
    BOOL a2; a2:=NOT (t OR f); tstb(112, FALSE, a2);
    BEGIN
        BOOL t; t := TRUE;
        BOOL f; f:=FALSE;
        IF (NOT (NOT ((t OR t) AND (t OR t)) OR
                ((f OR f) OR f) AND (t OR f AND f)) AND f)
                OR NOT t
        THEN tste(113) ELSE SKIP FI;
        BOOL x; x:=
            (NOT (NOT ((t OR t) AND (t OR t)) OR
            ((f OR f) OR f) AND (t OR (f AND f))) AND f)
            OR NOT t;
        tstb(114, FALSE, x);
        BOOL y =
            (NOT (NOT ((t OR t) AND (t OR t)) OR
            ((f OR f) OR f) AND (t OR (f AND f))) AND f)
            OR NOT t;
        tstb(115, FALSE, y)
    END
END;

print(("Test: NE, EQ for booleans", newline));
BEGIN
    BOOL t = TRUE; BOOL f = FALSE;
    tstb(116, TRUE, t NE f);
    tstb(117, FALSE, t NE t);
    tstb(118, TRUE, f NE t);
    tstb(119, FALSE, f NE f);
    tstb(120, TRUE, NOT (f NE f));
    tstb(121, FALSE, NOT (f EQ f));
    tstb(122, TRUE, t EQ t);
    tstb(123, FALSE, t EQ f);
    tstb(124, FALSE, f EQ t);
    tstb(125, TRUE, f EQ f);
    tstb(126, FALSE, (t NE t) NE (f NE f));
    tstb(127, TRUE, (t NE t) EQ (f NE f));
    tstb(128, TRUE, t NE (f NE f));
    tstb(129, TRUE, NOT (t EQ (f NE f)));
    tstb(130, FALSE, NOT NOT ((f NE f) NE f));
    tstb(131, FALSE, NOT NOT NOT ((f NE f) EQ f));
    IF t NE f THEN SKIP ELSE tste(132) FI;
    IF t NE t THEN tste(133) FI;
    IF f NE t THEN SKIP ELSE tste(134) FI;
    IF f NE f THEN tste(135) FI;
    IF NOT (f NE f) THEN SKIP ELSE tste(136) FI;
    IF NOT (f EQ f) THEN tste(137) FI;
```



```
    IF t EQ t THEN SKIP ELSE tste(138) FI;
    IF t EQ f THEN tste(139) FI;
    IF f EQ t THEN tste(140) FI;
    IF f EQ f THEN SKIP ELSE tste(141) FI;
    IF (t NE t) EQ (t NE t) THEN SKIP ELSE tste(142) FI;
    IF (t NE t) NE (t NE t) THEN tste(143) FI;
    IF (t NE t) NE f THEN tste(144) FI;
    IF (t NE t) EQ t THEN tste(145) FI;
    IF t NE (t NE t) THEN SKIP ELSE tste(146) FI;
    IF t EQ (t NE f) THEN SKIP ELSE tste(147) FI;
    BOOL a1; a1:=t EQ f; tstb(148, FALSE, a1);
    BOOL a2; a2:=NOT (t NE f); tstb(149, FALSE, a2);
    BEGIN
        BOOL t; t := TRUE;
        BOOL f; f:=FALSE;
        IF (NOT (NOT ((t NE t) EQ (t NE t)) NE
                ((f NE f) NE f) EQ (t NE f EQ f)) EQ f)
                NE NOT t
        THEN SKIP ELSE tste(150) FI;
        BOOL x; x:=
            (NOT (NOT ((t NE t) EQ (t NE t)) NE
            ((f NE f) NE f) EQ (t NE (f EQ f))) EQ f)
            NE NOT t;
        tstb(151, TRUE, x);
        BOOL y =
            (NOT (NOT ((t NE t) EQ (t NE t)) NE
            ((f NE f) NE f) EQ (t NE (f EQ f))) EQ f)
            NE NOT t;
        tstb(152, TRUE, y)
    END
END;

print(("Test: NOT", newline));
BEGIN
    BOOL a1, a2, a3, b1, b2, b3;
    BOOL a4=NOT FALSE; BOOL a5=NOT NOT FALSE;
    BOOL a6=NOT NOT NOT FALSE;
    BOOL a7=NOT a6; BOOL a8=NOT a7; BOOL a9=NOT NOT a8;
    BOOL a10=NOT NOT NOT a9;
    IF NOT TRUE THEN tste(153) FI;
    IF NOT NOT TRUE THEN SKIP ELSE tste(154) FI;
    IF NOT NOT NOT TRUE THEN tste(155) FI;
    a1:=NOT TRUE; a2:=NOT NOT FALSE;
    a3:=NOT NOT NOT TRUE;
    b1:=NOT TRUE AND FALSE;
    b2:=NOT NOT TRUE AND FALSE;
    b3:=TRUE OR NOT NOT NOT TRUE OR FALSE;
    tstb(156, FALSE, a1);
    tstb(157, FALSE, a2);
    tstb(158, FALSE, a3);
    tstb(159, TRUE, a4);
```

```
    tstb(160, FALSE, a5);
    tstb(161, TRUE, a6);
    tstb(162, FALSE, a7);
    tstb(163, TRUE, a8);
    tstb(164, TRUE, a9);
    tstb(165, FALSE, a10);
    tstb(166, FALSE, b1);
    tstb(167, FALSE, b2);
    tstb(168, TRUE, b3)
END;

print(("Test: EQ, NE, LT, LE, GT, GE", newline));
BEGIN
    IF -1=-1 THEN SKIP ELSE tste(169) FI;
    tstb(170, TRUE, -LONG 1=-LONG 1);
    IF -1/=1 THEN SKIP ELSE tste(171) FI;
    tstb(172, TRUE, -LONG 1/=LONG 1);
    IF 1 /= -1 THEN SKIP ELSE tste(173) FI;
    tstb(174, TRUE, LONG 1/=-LONG 1);
    IF 1 =1 THEN SKIP ELSE tste(175) FI;
    tstb(176, TRUE, LONG 1=LONG 1);
    IF 0=0.0 THEN SKIP ELSE tste(177) FI;
    tstb(178, TRUE, LONG 0=LONG 0.0);
    IF 1.0=1 THEN SKIP ELSE tste(179) FI;
    tstb(180, TRUE, LONG 1.0=LONG 1);
    IF -1.0=-1.0 THEN SKIP ELSE tste(181) FI;
    tstb(182, TRUE, -LONG 1.0=-LONG 1.0);
    IF -1.0/=1.0 THEN SKIP ELSE tste(183) FI;
    tstb(184, TRUE, LENG-1.0/=LONG 1.0);
    IF 1.0/= -1.0 THEN SKIP ELSE tste(185) FI;
    tstb(186, TRUE, LONG 1.0/=LENG-1.0);
    BOOL
    a1=1=1, a2=1/=1, a3=1>1,
    a4=1<1, a5=1<= 1, a6=1>=0,
    a7=1.0=2.0, a8=1.0/=2.0, a9=1.0<2.0,
    a10=1.0>0.0, a11=1.0<=1.0,
    a12=1.0>=-1.0;
    tstb(187, TRUE, a1);
    tstb(188, FALSE, a2);
    tstb(189, FALSE, a3);
    tstb(190, FALSE, a4);
    tstb(191, TRUE, a5);
    tstb(192, TRUE, a6);
    tstb(193, FALSE, a7);
    tstb(194, TRUE, a8);
    tstb(195, TRUE, a9);
    tstb(196, TRUE, a10);
    tstb(197, TRUE, a11);
    tstb(198, TRUE, a12);
    IF 1 = 1 THEN SKIP ELSE tste(199) FI;
    tstb(200, TRUE, 1=1);
```



```
    IF NOT (1/=1) THEN SKIP ELSE tste(201) FI;
    tstb(202, TRUE, NOT NOT NOT (1/=1));
    IF 1/=2 AND 2/=3 AND 4/=5 THEN SKIP ELSE tste(203) FI
END;

BEGIN
    REAL j;
    [-3 : 3] BOOL lt0;
    lt0[-3]:= lt0[-2]:= lt0[-1]:= TRUE;
    lt0[0]:= lt0[1]:= lt0[2]:= lt0[3]:= FALSE;
    FOR i FROM -3 BY 1 TO 3
    DO
        tstb(204, NOT(lt0[i] OR lt0[-i]), i=0);
        tstb(205, lt0[i] OR lt0[-i], i/=0);
        tstb(206, lt0[-i], i>0);
        tstb(207, NOT lt0[i], i>=0);
        tstb(208, lt0[i], i<0);
        tstb(209, NOT lt0[-i], i<=0);
        j:=i;
        tstb(210, NOT(lt0[i] OR lt0[-i]), j=0);
        tstb(211, lt0[i] OR lt0[-i], j/=0);
        tstb(212, lt0[-i], j>0);
        tstb(213, NOT lt0[i], j>=0);
        tstb(214, lt0[i], j<0);
        tstb(215, NOT lt0[-i], j<=0)
    OD
END;

print(("Test: monadic -", newline));
BEGIN
    INT x0, x1, x2, x3; LONG INT z0, z1, z2, z3;
    REAL y0, y1, y2, y3; LONG REAL t0, t1, t2, t3;
    INT x4 = 10; LONG INT z4 = LONG 10;
    REAL y4 = x4; LONG REAL t4 = z4;
    z0 := --LONG 38; z1 := -LONG 1000000000; z2 := -z1;
    x0:=--79; x1 := -1; x2 := -x1; x3 := -SHORTEN z0;
    t0 := --LONG 8.7; t1 := -LONG 79.99e-2; t2 := -t1;
    y0 := --6.7e-4; y1 := -39.47e-2; y2 := -y1; y3 := -SHORTEN t2
    tsti(216, +79, x0);
    tsti(217, -1, x1);
    tsti(218, +1, x2);
    tsti(219, -38, x3);
    tsti(220, +10, x4);
    tstli(221, +LONG 38, z0);
    tstli(222, -LONG 1000000000, z1);
    tstli(223, +LONG 1000000000, z2);
    tstli(224, +LONG 10, z4);
    tstr(225, +6.7e-4, y0);
    tstr(226, -3.947e-1, y1);
    tstr(227, +3.947e-1, y2);
    tstr(228, -7.999e-1, y3);
```

```
    tstr(229, +1e+1, y4);
    tstlr(230, +LONG 8.7e+0, t0);
    tstlr(231, -LONG 7.999e-1, t1);
    tstlr(232, +LONG 7.999e-1, t2);
    tstlr(233, +LONG 10e+0, t4)
END;

print(("Test: dyadic -", newline));
BEGIN
    tsti(234, -2, 1-3);
    tstli(235, +LONG 4, LONG 7-LONG 3);
    tsti(236, -4, -1-3);
    tstli(237, -LONG 13, -LONG 8-LONG 5);
    tsti(238, +4, 1--3);
    tstli(239, +LONG 7, LONG 4--LONG 3);
    tsti(240, +2, -1--3);
    tstli(241, -LONG 10, -LONG 5--LENG-5);

    tstr(242, -6.4e+0, 1-7.4);
    tstlr(243, +LONG 8e+0, LONG 4-LENG-4.0);
    tstr(244, +6.4e+0, 7.4-1);
    tstlr(245, -LONG 1.2e+1, -LONG 6.0-LONG 6);
    tstr(246, +1.22e+1, 6.1--6.1);
    tstlr(247, +LONG 8.6e+0, LONG 4.3--LONG 4.3);

    tsti(248, +29999, 30000-1);
    tstli(249, +LONG 999999998, LONG 1000000000-LONG 2);

    tstr(250, +1e+0, -1.0-(-1.0-(-1.0-(-1.0-1))));
    tstlr(251, +LONG 0e+0,
        -LONG 1.0-(-LONG 1.0-(-LONG 1.0-(-LONG 1.0
        -(LONG 1-LONG 1.0)))))
END;

print(("Test: dyadic +", newline));
BEGIN
    tsti(252, +4, 1+3);
    tstli(253, +LONG 10, LONG 7+LONG 3);
    tsti(254, +2, -1+3);
    tstli(255, -LONG 3, -LONG 8+LONG 5);
    tsti(256, -2, 1+-3);
    tstli(257, +LONG 1, LONG 4+-LONG 3);
    tsti(258, -4, -1+-3);
    tstli(259, +LONG 0, -LONG 5+-LENG-5);

    tstr(260, +8.4e+0, 1+7.4);
    tstlr(261, +LONG 0e+0, LONG 4+LENG-4.0);
    tstr(262, +8.4e+0, 7.4+1);
    tstlr(263, +LONG 0e+0, -LONG 6.0+LONG 6);
    tstr(264, +0e+0, 6.1+-6.1);
    tstlr(265, +LONG 0e+0, LONG 4.3+-LONG 4.3);
```



```
    tsti(266, -29999, -30000+1);
    tstli(267, -LONG 999999998, -LONG 1000000000+LONG 2);

    tstr(268, -3e+0, -1.0+(-1.0+(-1.0+(-1.0+1))));
    tstlr(269, -LONG 2e+0,
        -LONG 1.0+(-LONG 1.0+(-LONG 1.0+(-LONG 1.0
        +(LONG 1+LONG 1.0)))))
END;

print(("Test: ABS", newline));
BEGIN
    tsti(270, +19, ABS 19);
    tsti(271, +19, ABS-19);
    tsti(272, +0, ABS 0);
    tsti(273, +32, SHORTEN ABS LONG 32);
    tsti(274, +43, SHORTEN ABS-LONG 43);
    tsti(275, +0, SHORTEN ABS LONG 0);
    tstr(276, +1.97e+2, ABS 197.0);
    tstr(277, +4.97e+1, ABS-49.7);
    tstr(278, +0e+0, ABS .0);
    tstr(279, +1e+0, SHORTEN ABS LONG 1.0);
    tstr(280, +0e+0, SHORTEN ABS LONG 0.0e+7);
    tstr(281, +1.9e+0, SHORTEN ABS-LONG 1.9)
END;

print(("Test: * for integers", newline));
BEGIN
    tsti(282, +6, 2*3);
    tsti(283, -6, -2*3);
    tsti(284, -6, 2*-3);
    tsti(285, +6, -2*-3);
    tsti(286, +6, SHORTEN(-LONG 2*-LONG 3));
    tsti(287, -6, SHORTEN(LONG 2*-LONG 3));
    tsti(288, -6, SHORTEN(-LONG 2*LONG 3));
    tsti(289, +6, SHORTEN(LONG 2*LONG 3));
    tsti(290, +0, 0*10);
    tstli(291, +LONG 0, LONG 0*-LONG 10);
    tsti(292, +0, 10*0);
    tstli(293, +LONG 0, -LONG 10*LENG 0);
    tsti(294, +3, 1*3);
    tstli(295, +LONG 3, LONG 3*LONG 1);
    tsti (296, +16384, 2*(2*(2*(2*(2*(2*(2*(2*(2*(2*(2*(2*(2*(2
        ))))))))))))))
END;

print(("Test: * mixed", newline));
BEGIN
    REAL a1=1.0; LONG REAL a2=-LONG 1.0;
    REAL a3; LONG REAL a4;
    tstr(297, +4.2e+1, 6.0*7.0);
    tstlr(298, +LONG 2.6e+1, LONG 13.0*LONG 2.0);
```

```
    tstr(299, -4.2e+1, -6.0*7.0);
    tstlr(300, -LONG 6e+0, -LONG 2.0*LONG 3.0);
    tstr(301, -5.6e+1, 7.0*-8.0);
    tstlr(302, -LONG 4.8e+1, LONG 16.0*-LENG 3.0);
    tstr(303, +5.6e+1, -8.0*-7.0);
    tstlr(304, +LONG 2.5e+1, -LENG 5.0*-LONG 5.0);

    tstr(305, +4.2e+1, 7*6.0);
    tstlr(306, +LONG 2.1e+1, LONG 3*LONG 7.0);
    tstr(307, +4.2e+1, 6.0*7);
    tstlr(308, +LONG 3.6e+1, LONG 6.0*LONG 6);
    tstr(309, +4.2e+1, -7*-6.0);
    tstlr(310, +LONG 2.25e+2, -LONG 15*-LONG 15.0);
    tstr(311, +4.2e+1, -6.0*-7);
    tstlr(312, +LONG 1.9e+1, -LONG 19.0*-LONG 1);

    a3:=14.0; a4:=LENG-13.0;
    tstr(313, +1.4e+1, a3*a1);
    tstlr(314, +LONG 1.3e+1, a2*a4);
    tstr(315, +10e-43, 1.0e-20*1.0e-22);
    tstlr(316, +LONG 10e+39, LONG 1.0e+20*LONG 1.0e+20)
END;

print(("Test: OVER, MOD", newline));
BEGIN
    tsti(317, +2, 12 OVER 6);
    tsti(318, -5, -20 OVER 4);
    tsti(319, -25, 100 OVER -4);
    tsti(320, +10, -10 OVER -1);
    tsti(321, +2, 7 OVER 3);
    tsti(322, -2, -8 OVER 3);
    tsti(323, -1, 10 OVER -7);
    tsti(324, +1, -49 OVER -27);
    tsti(325, +3, SHORTEN(LONG 10 OVER LONG 3));
    tsti(326, -1, SHORTEN(-LONG 50 OVER LONG 50));
    tsti(327, -1, -12 OVER 7);
    tsti(328, -1, 12 OVER -7);
    tsti(329, -1, SHORTEN(-LONG 50 OVER LONG 50));
    tsti(330, +0, SHORTEN(LONG 0 OVER -LONG 25));
    tsti(331, +0, 12 MOD 6);
    tsti(332, +0, -20 MOD 4);
    tsti(333, +0, 100 MOD -4);
    tsti(334, +0, -10 MOD -1);
    tsti(335, +1, 7 MOD 3);
    tsti(336, +1, -8 MOD 3);
    tsti(337, +3, 10 MOD -7);
    tsti(338, +5, -49 MOD -27);
    tsti(339, +1, SHORTEN(LONG 10 MOD LONG 3));
    tsti(340, +0, SHORTEN(-LONG 50 MOD LONG 50));
    tsti(341, +2, -12 MOD 7);
    tsti(342, +5, 12 MOD -7);
```



```
    tsti(343, +0, SHORTEN(-LONG 50 MOD LONG 50));
    tsti(344, +0, SHORTEN(LONG 0 MOD -LONG 25))
END;

print(("Test: /", newline));
BEGIN
    REAL a, b; LONG REAL c, d; REAL x = 127.0;
    FOR i FROM -3 BY 1 TO 3 DO
    FOR j FROM -3 BY 1 TO 3 DO
        IF j /= 0 THEN
            a:=i/j;
            tstr(345, i, a*j);
            a:=i; a:=a/j;
            tstr(346, i, a*j);
            a:=j; a :=i/a;
            tstr(347, i, a*j);
            a := i; b := j; a := a/b;
            tstr(348, i, a*j);
            c := LENG i/LENG j;
            tstlr(349, LENG i, c*LENG j);
            c := LENG i; c := c/LENG j;
            tstlr(350, LENG i, c*LENG j);
            c := LENG j; c :=LENG i/c;
            tstlr(351, LENG i, c*LENG j);
            c := LENG i; d:=LENG j; c:=c/d;
            tstlr(352, LENG i, c*LENG j)
        FI
    OD OD;
    tstr(353, -1e+0, 1.9e-7/-1.9e-7);
    a:=19.74e+2;
    tstr(354, +1e+0, 19.74e+2/a);
    tstr(355, +10e-5, 19.74e-2/19.74e+2);
    tstr(356, +1e+3, 127000.0/x);
    tstr(357, +1e+2, x/1.27);
    tstr(358, -1e+1, x/-12.7);
    tstr(359, +1e+0, x/x);
    a:=1270.0;
    tstr(360, +10e-2, x/a);
    a:=0.0149; tstr(361, +1e+0, a/149e-4);
    tstr(362, +1e+0, a/a);
    tstr(363, +1.173228346456693e-4, a/x);
    tstr(364, -1e-2, a/-1.49);
    tstr(365, +1.27e+2, -x/-1.0);
    tstr(366, -1e+0, -x/x);
    tstr(367, -8.523489932885906e+3, -x/a);
    tstr(368, +1e+0, -x/-x)
END;

print(("Test: **", newline));
BEGIN
    INT a;
```

```
    tsti(369, +1, 1**0);
    tsti(370, +1, 10**0);
    tsti(371, +1, -20**0);
    tsti(372, +1, 1**1);
    tsti(373, +10, 10**1);
    tsti(374, -10, -10**1);
    tsti(375, +0, 0**1);
    tsti(376, +0, 0**30000);
    tsti(377, +1, 0**0);
    tsti(378, +49, 7**2);
    tsti(379, +1, 1**2);
    tsti(380, +9, -3**2);
    tsti(381, -27, -3**3);
    tsti(382, +32, 2**5);
    tsti(383, +81, SHORTEN(LONG 9**2));
    tsti(384, -19683, SHORTEN(-LONG 27**3));
    tstr(385, +2.7e+1, 3.0**3);
    tstr(386, +6.5536e+4, 2.0**16);
    tstr(387, -3.2768e+4, -2.0**15);
    tstr(388, +3.6e+1, -6.0**2);
    tstr(389, +2.5e-1, 2.0**-2);
    tstr(390, -10e-4, -10.0**-3);
    tstr(391, +1e+0, 3.0**-0);
    tstr(392, +1e+0, 3.0**-0);
    tstr(393, +4.9e+1, SHORTEN(LONG 7.0**2));
    tstr(394, +3.969e+1, SHORTEN(-LONG 6.3**2));
    tstr(395, +1.385019350059107e-8, SHORTEN(LONG 37.3**-5));
    a:= 1;
    FOR i FROM 1 BY 1 TO 10
    DO  tsti(396, a, (-1)**(i-1)); a:= -a OD;
    a:= 0;
    FOR i FROM 1 BY 1 TO 10
    DO a := a + 1**30000 OD;
    tsti(397, +10, a)
END;

print(("Test: SHORTEN, LENG", newline));
BEGIN
    LONG INT a1 = LONG 128; LONG INT a2;
    LONG REAL a3 = LONG 1.9999999999; LONG REAL a4;
    REAL a6;
    INT a5; a5 := 30000;
    a6 := 2/3;
    tstli(398, +LONG 179, LENG 179);
    tsti(399, +19, SHORTEN LONG 19);
    tsti(400, +30000, SHORTEN LONG 30000);
    tsti(401, -27, SHORTEN-LONG 27);
    tsti(402, -30000, SHORTEN-LONG 30000);
    tsti(403, +128, SHORTEN a1);
    a2:=LONG 0;
    tsti(404, +0, SHORTEN a2);
```



```
    tsti(405, +30000, SHORTEN LENG a5);
    tstr(406, +1.234566666e-1, SHORTEN LONG .1234566666);
    tstr(407, +1.9999999999e+0, SHORTEN a3);
    a4:=LONG .1111111111; tstr(408, +1.11111111le-1, SHORTEN a4);
    tstr(409, -3.333333333333333e-1, SHORTEN-LENG (1/3));
    tstr(410, +6.666666666666667e-1, SHORTEN LENG a6)
END;

print(("Test: ODD", newline));
BEGIN
    IF ODD-1 THEN SKIP ELSE tste(411) FI;
    tstb(412, FALSE, ODD 2);
    IF NOT ODD-LONG 2 THEN SKIP ELSE tste(413) FI;
    tstb(414, TRUE, ODD LONG 1);
    BOOL a1, a2;
    a1:= FALSE;
    FOR i FROM -10 BY 1 TO 10
    DO tstb(415, a1, ODD i); a1:= NOT a1 OD;
    a1:=ODD-3;
    a2:=ODD-LONG 0;
    BOOL b1 = NOT ODD -13;
    BOOL b2 = NOT NOT ODD -LONG 16;
    tstb(416, TRUE, a1);
    tstb(417, FALSE, a2);
    tstb(418, FALSE, b1);
    tstb(419, FALSE, b2);
    tstb(420, FALSE, NOT NOT NOT ODD 55);
    tstb(421, FALSE, NOT NOT NOT ODD LONG 1)
END;

print(("Test: SIGN", newline));
BEGIN
    tsti(422, +1, SIGN 7);
    tsti(423, +0, SIGN 0);
    tsti(424, -1, SIGN-7);
    tsti(425, +1, SIGN LONG 1000000000);
    tsti(426, +0, SIGN LONG 0);
    tsti(427, -1, SIGN-LONG 8);
    tsti(428, +1, SIGN 1.9);
    tsti(429, +0, SIGN 0.0);
    tsti(430, -1, SIGN-3.6);
    tsti(431, +1, SIGN LONG 67.0);
    tsti(432, +0, SIGN LONG 0.0);
    tsti(433, -1, SIGN-LONG 37.0)
END;

print(("Test: ROUND, ENTIER", newline));
BEGIN
    REAL a1 = 1.7; LONG REAL a2 = LONG 27.7;
    REAL a3; LONG REAL a4;
    tstli(434, -LONG 28, LENG ROUND-27.7);
```

```
    tstli(435, -LONG 28, LENG ENTIER-27.7);
    tstli(436, +LONG 2, LENG ROUND a1);
    tstli(437, +LONG 1, LENG ENTIER a1);
    tstli(438, +LONG 13, LENG ROUND 12.9);
    tstli(439, +LONG 12, LENG ENTIER 12.994);
    a3:=134e+2; a4:=LONG 135.1e-1;
    tstli(440, +LONG 13400, LENG ROUND a3);
    tstli(441, +LONG 13400, LENG ENTIER a3);
    tstli(442, +LONG 14, LENG ROUND SHORTEN a4);
    tstli(443, +LONG 13, LENG ENTIER SHORTEN a4);
    tstli(444, -LONG 1, ROUND-LENG 127e-2);
    tstli(445, +LONG 1, ENTIER--LONG 127e-2);
    tstli(446, +LONG 28, ROUND a2);
    tstli(447, +LONG 27, ENTIER a2);
    tstli(448, +LONG 13, ROUND LONG 12.87);
    tstli(449, +LONG 12, ENTIER LONG 12.87);
    tstli(450, +LONG 14, ROUND a4);
    tstli(451, +LONG 13, ENTIER a4);
    tstli(452, -LONG 2, ROUND-LONG 1.5001);
    tstli(453, -LONG 2, ENTIER-LENG 1.5001);
    tstli(454, +LONG 6, LENG ROUND 6.499)
END;

print(("Test: EQ, NE, LT, LE, GT, GE for chars", newline));
BEGIN
    CHAR a = "1"; CHAR b = "2"; CHAR c; c := "1";
    [1:1]CHAR d; d[1]:="$";
    INT abs0 = ABS"0", abs1 = ABS"1", abs2 = ABS"2";

    tstb(455, TRUE, "1"="1");
    tstb(456, FALSE, "1"/="1");
    tstb(457, TRUE, "1"<="1");
    tstb(458, FALSE, "1"<"1");
    tstb(459, TRUE, "1">="1");
    tstb(460, FALSE, "1">"1");
    tstb(461, FALSE, "1"="2");
    tstb(462, TRUE, "1"/="2");
    tstb(463, TRUE, "1"<="2");
    tstb(464, TRUE, "1"<"2");
    tstb(465, FALSE, "1">="2");
    tstb(466, FALSE, "1">"2");
    tstb(467, FALSE, "2"="1");
    tstb(468, TRUE, "2"/="1");
    tstb(469, FALSE, "2"<="1");
    tstb(470, FALSE, "2"<"1");
    tstb(471, TRUE, "2">="1");
    tstb(472, TRUE, "2">"1");

    tstb(473, TRUE, a=a);
    tstb(474, FALSE, a/=a);
    tstb(475, TRUE, a<=a);
```



```
    tstb(476, FALSE, a<a);
    tstb(477, TRUE, a>=a);
    tstb(478, FALSE, a>a);
    tstb(479, FALSE, a=b);
    tstb(480, TRUE, a/=b);
    tstb(481, TRUE, a<=b);
    tstb(482, TRUE, a<b);
    tstb(483, FALSE, a>=b);
    tstb(484, FALSE, a>b);
    tstb(485, FALSE, b=a);
    tstb(486, TRUE, b/=a);
    tstb(487, FALSE, b<=a);
    tstb(488, FALSE, b<a);
    tstb(489, TRUE, b>=a);
    tstb(490, TRUE, b>a);

    tstb(491, TRUE, REPR abs1=REPR abs1);
    tstb(492, FALSE, REPR abs1/=REPR abs1);
    tstb(493, TRUE, REPR abs1<=REPR abs1);
    tstb(494, FALSE, REPR abs1<REPR abs1);
    tstb(495, TRUE, REPR abs1>=REPR abs1);
    tstb(496, FALSE, REPR abs1>REPR abs1);
    tstb(497, FALSE, REPR abs1=REPR abs2);
    tstb(498, TRUE, REPR abs1/=REPR abs2);
    tstb(499, TRUE, REPR abs1<=REPR abs2);
    tstb(500, TRUE, REPR abs1<REPR abs2);
    tstb(501, FALSE, REPR abs1>=REPR abs2);
    tstb(502, FALSE, REPR abs1>REPR abs2);
    tstb(503, FALSE, REPR abs2=REPR abs1);
    tstb(504, TRUE, REPR abs2/=REPR abs1);
    tstb(505, FALSE, REPR abs2<=REPR abs1);
    tstb(506, FALSE, REPR abs2<REPR abs1);
    tstb(507, TRUE, REPR abs2>=REPR abs1);
    tstb(508, TRUE, REPR abs2>REPR abs1);

    tstb(509, TRUE, REPR abs1=c);
    tstb(510, FALSE, REPR abs1/=c);
    tstb(511, TRUE, REPR abs1<=c);
    tstb(512, FALSE, REPR abs1<c);
    tstb(513, TRUE, REPR abs1>=c);
    tstb(514, FALSE, REPR abs1>c);
    tstb(515, FALSE, REPR abs0=c);
    tstb(516, TRUE, REPR abs0/=c);
    tstb(517, TRUE, REPR abs0<=c);
    tstb(518, TRUE, REPR abs0<c);
    tstb(519, FALSE, REPR abs0>=c);
    tstb(520, FALSE, REPR abs0>c);
    tstb(521, FALSE, REPR abs2=c);
    tstb(522, TRUE, REPR abs2/=c);
    tstb(523, FALSE, REPR abs2<=c);
    tstb(524, FALSE, REPR abs2<c);
```

```
    tstb(525, TRUE, REPR abs2>=c);
    tstb(526, TRUE, REPR abs2>c);

    tstb(527, TRUE, c=REPR abs1);
    tstb(528, FALSE, c/=REPR abs1);
    tstb(529, TRUE, c<=REPR abs1);
    tstb(530, FALSE, c<REPR abs1);
    tstb(531, TRUE, c>=REPR abs1);
    tstb(532, FALSE, c>REPR abs1);
    tstb(533, FALSE, c=REPR abs0);
    tstb(534, TRUE, c/=REPR abs0);
    tstb(535, FALSE, c<=REPR abs0);
    tstb(536, FALSE, c<REPR abs0);
    tstb(537, TRUE, c>=REPR abs0);
    tstb(538, TRUE, c>REPR abs0);
    tstb(539, FALSE, c=REPR abs2);
    tstb(540, TRUE, c/=REPR abs2);
    tstb(541, TRUE, c<=REPR abs2);
    tstb(542, TRUE, c<REPR abs2);
    tstb(543, FALSE, c>=REPR abs2);
    tstb(544, FALSE, c>REPR abs2);

    tstb(545, TRUE, "$"=d[1]);
    tstb(546, FALSE, "$"/=d[1]);
    tstb(547, TRUE, "$"<=d[1]);
    tstb(548, FALSE, "$"<d[1]);
    tstb(549, TRUE, "$">=d[1]);
    tstb(550, FALSE, "$">d[1]);
    tstb(551, TRUE, "$"=d[1])
END

END
```

- - . - -

```
#idef01#
11: IF INT i:=1; FALSE THEN INT i:= 2; print(i)      ELSE
  print(i) #1# FI
```

- - . - -

```
#idef02#
12: IF INT i:=1; TRUE THEN INT i:= 2; print(i) #2# ELSE
  print(i)      FI
```

- - . - -


```
#idef03#
BEGIN INT i = 1;

  PROC a = VOID : ( INT i = 2; b);
  PROC b = VOID : print(i);

  a # +1 #
END
```

- - . - -

```
#idef04#
BEGIN INT i = 1, j = -1;

  PROC a = VOID : (INT i = 2, j = -2; b);

  PROC b = VOID :
    ( INT j = -3; PROC c = VOID: print(i + j); d(c));

  PROC d = (PROC VOID e) VOID :
    ( INT i = 4, j = -4; e);

  a # -2 #
END
```

- - . - -

```
#idef05#
BEGIN INT i:= 1; (INT i = i; print(i) # what is the value of i? #)
END
```

- - . - -

```
#idef06#
BEGIN # Operators #

  STRING int = "INT   ", real = "REAL  ", rreal = "[]REAL";
  PROC(REF FILE)VOID n = newline;

  print(("Results must be:", n,
    int, 1, n, real, 1.0, n, rreal, 1.0, n, real, 3.0, n, rreal, 3.0,
    n, rreal, 2.0, n, rreal, 2.0, n, int, 3, n, real, 3.0, n,
    rreal, 3.0, n, real, 4.0, n, rreal, 4.0, n, real, 4.0, n,
    rreal, 4.0, n, n,
    "Results are:", n));

  OP AA = (UNION(INT, REAL, []REAL) p)
    UNION(REAL, []REAL):
```

```
  CASE p
  IN
    (INT i): (print((int, i, n)); AA REAL(i)),
    (REAL r): (print((real, r, n)); AA []REAL(r))
  OUSE print((rreal, p, n)); p
  IN
    ([]REAL rr):
      CASE ROUND rr[1] IN 3.0, rr OUT 4.0 ESAC
  OUT error
  ESAC;

  FOR i TO 3
  DO
    AA AA
      CASE i IN
        UNION(REAL, INT)(1),
        UNION(INT, []REAL) ([]REAL(2)),
        AA 3
      ESAC
  OD

EXIT error: print("Error in united-case-clause")

END
```

- - . - -

```
#idef07#
BEGIN # Redeclaring LWB #

  OP LWB = ([]INT  a) REAL : a[1] + a[2];
  OP LWB = ([]REAL a) REAL : a[1] - a[2];

  print(LWB (1| (8, 2), 3, []INT : SKIP));  # 10.0 #
  print(LWB (1| (8, 2), 3, []REAL: SKIP));  #  6.0 #

  SKIP
END
```

- - . - -

```
#idef08#
BEGIN # Hiding of operators #

  print(("Should not run", newline));

  # To be hidden: #
  OP + = (UNION(INT, REAL, BOOL) p) INT : 2;
```



```
  (OP + = (INT i) INT : 3;                    # hides #
    print(+ 1); print(+ 1.0)  # OK, KO #; print(newline));

  (OP + = (REF PROC REAL i ) INT : 3;         # hides #
    print(+ 1); print(+ 1.0)  # KO, KO #; print(newline));

  (OP + = ([]REAL i ) INT : 3;                # does not hide #
    print(+ 1); print(+ 1.0)  # OK, OK #; print(newline));

  (OP + = (UNION([]INT, []REAL) i ) INT : 3;
                                              # does not hide #
    print(+ 1); print(+ 1.0)  # OK, OK #; print(newline));

  (OP + = (UNION([]INT, REAL) i ) INT : 3;    # hides #
    print(+ 1); print(+ 1.0)  # KO, OK #; print(newline));

  (OP + = (REF UNION(INT, BOOL) i ) INT : 3; # hides #
    print(+ 1); print(+ 1.0)  # KO, KO #; print(newline));

  (OP + = (UNION(CHAR, REF UNION(INT, BOOL)) i ) INT : 3;
                                              # hides #
    print(+ 1); print(+ 1.0)  # KO, KO #; print(newline));

  (OP + = (UNION(CHAR, REF UNION(REF INT, REF BOOL)) i )
              INT : 3;                        # does not hide #
    print(+ 1); print(+ 1.0)  # OK, OK #; print(newline));

  SKIP
END
```

- - . - -

```
#idef09#
( # Obscuring LWB and UPB #

  print(("Should not run",newline));

  (OP LWB = (INT i) INT : 1;
    LWB []REAL(1)     # OK # );

  (OP UPB = ([]INT i) INT : 1;
    UPB []REAL(1)     # KO # );

  (OP LWB = (REF[]INT i) INT : 1;
    LWB []REAL(1)     # KO # );

  (OP UPB = (REF UNION([]INT, []BOOL) i) INT : 1;
    UPB []REAL(1)     # KO # );

  (OP LWB = (REF UNION(REF[]INT, []BOOL) i) INT : 1;
```

```
    LWB []REAL(1)      # OK # );

  (OP UPB = (UNION(REF[]INT, []BOOL) i) INT : 1;
    UPB []REAL(1)      # KO # );

  SKIP
)
```

- - . - -

```
#idef10#
BEGIN FOR i FROM 1 BY i DO SKIP OD
  # second 'i' is unknown #
END
```

- - . - -

```
#idef11#
BEGIN # More operators and uniting #

  MODE UN = UNION (INT, REAL);

  OP + = (UNION (REF UN, REF CHAR) a) VOID:
    ( CASE a IN
        (REF UN ru):
          (ru |
            (INT i): print(("integer", i)),
            (REAL r): print(("real   ", r))
          ),
        (REF CHAR ch): print(("char   ", ch))
      ESAC;

      print(newline)
    );

  +(HEAP UN:= 1);
  +(HEAP UN:= 2.0);
  +(HEAP CHAR:= "3")
END
```

- - . - -

```
#idef12#
  # Priorities and weird constructions #
WHILE INT n:= 0;
    OP + = (REF INT i, CHAR c) STRING: "ab";
  ( WHILE DO GOTO skip OD; TRUE DO SKIP OD;
    PRIO + = 1; SKIP
```



```
    EXIT skid:
        (HEAP INT +:= 1 + "1" +=: (HEAP STRING := "c")) = "abc"
    EXIT skip: GOTO skid
  )
DO DO
    # The implicit structure of the formulas is
            ( a O1 ( b + ( c O2 d)))
      which is only achieved if   pr(O1) < pr(+) < pr(O2)
    #

    PRIO + = 2;

    OP + = (INT i, BOOL b) STRING: (print("corr"); "ect,");
    OP + = (REAL x, BOOL b) BOOL: (print("Line "); FALSE);
    OP + = (CHAR c, BOOL b) BOOL: (print("two "); FALSE);
    OP + = (BITS b, INT i) STRING: (print("one "); "Nope ");
    OP + = (REF BYTES b, REAL x) INT: (print("is "); -(n+:=1));
    OP + = (STRING s, COMPL c) BITS: (print("sho"); drop);

     print((HEAP STRING +:=
         ABS
         IF PRIO + = 3;
             ODD n OR 2.0 + "a" = "b"
         THEN PRIO + = 4;
             TRUE AND "a" + 2 < 3
         ELSE PRIO + = 5;
             "prio" = 2r1 + 2 - 3
         FI

         +              # prio 2 #

         CASE PRIO + = 7;
             0 - LOC BYTES + 3.0 ** 5
         IN TRUE, FALSE
         OUT PRIO + = 8;
             3 ELEM "prio" + 2.0 I 3.0
         ESAC

         AND random < .5,

         newline))
     OD
     EXIT drop:
         print(("rter than line three.", newline, "End of test"))
         stop
OD
```

- - . - -

```
#clau01#
BEGIN # Some routines #

  PROC p = (REALx) REAL:x+1;

  [ # 1 : 9 # ] UNION(PROC REAL, PROC(REAL)REAL)a=
   (sin, cos, REAL:3, (REAL x) REAL:x**2, p,
                      PROC REAL : REAL: 3.14,
    REAL:p(2), random, SKIP);

  FOR i TO UPB a DO
  print(CASE a[i] IN
   (PROC REAL pr): pr,
   (PROC(REAL) REAL pr):pr(i) OUT "skip"
   ESAC) OD

  # Output: +0. 841 470 984 807 5, -0. 416 146 836 546 4,
    3.0, 16.0, 6.0, 3.14, 3.0, some random number, skip #
END
```

- - . - -

```
#clau02#
BEGIN # Case conformity #
  MODE M = UNION ([]INT, BOOL, STRING);
  PROC prpm = REF PROC M: HEAP PROC M:= M : "aap ";

  FOR n TO 4 DO
  CASE CASE n IN TRUE, IF FALSE THEN "aa" ELSE
    "b " FI, prpm OUT LOC[1:1]INT:=1 ESAC
      IN (UNION(STRING, BOOL) sb): print(("sb ", sb)),
        ([]INT i): print(("i ", i))
      OUT print("void")
  ESAC OD

        # sb TRUE sb b sb aap i 1 #

END
```

- - . - -

```
#clau03#
BEGIN # Wrong clauses #

  INT i:= 1, BOOL b:= TRUE;
  UNION (INT, BOOL, REAL) ibr = SKIP;

  print(( i | 1, 2 |: i | 3, 4));                    #OK#
  print(( i | 1  2 |: i | 3, 4));                    #KO#
```



```
  print(( b | 1  2 |: i | 3, 4));                    #K
  print(( b | 1  2 |: b | 3, 4));                    #K
  print(( b | 1  2 |: b | 3  4));                    #O
  print(( i | 1, 2 |: b | 3  4));                    #K
  print(( i | 1, 2 |( b | 3  4)));                   #O

  print(( ibr | (INT): 1, (BOOL): 2 |: ibr | (REAL): 3 | 4));  #O
  print(( ibr | (INT): 1,         2 |: ibr | (REAL): 3 | 4));  #K
  print(( ibr |         1, (BOOL): 2 |: ibr | (REAL): 3 | 4));  #K
  print(( ibr | (INT): 1, (BOOL): 2 |: ibr |         3 | 4));  #K
  print(( ibr | (INT): 1, (BOOL): 2 |:  b  |         3 | 4));  #K
  print(( ibr | (INT): 1, (BOOL): 2 |(  b  |         3 | 4))); #O

  print(CASE ibr IN (INT):1, (BOOL):2 OUT 3 ESAC);          #OK#
  print(CASE  1  IN (INT):1, (BOOL):2 OUT 3 ESAC);          #KO#
  print(CASE "a" IN (INT):1, (BOOL):2 OUT 3 ESAC);          #KO#

  CASE CASE ibr IN (UNION (INT, BOOL) ib) : ib ESAC
  IN (BOOL) : ibr
  ESAC;                                                     #O

  CASE CASE ibr IN (UNION (INT, REAL) ir) : ir ESAC
  IN (BOOL) : ibr
  ESAC;                                                     #K

  SKIP
END
```

- - . - -

```
#clau04#
BEGIN # Vacuum #
  print(LWB[]INT BEGIN END  ); #1#
  print(UPB[]INT()); #0#
  print(UPB([]INT())[1:0]); #0#
  print(2UPB[,]INT([]INT(print("here ");()))); #0#
  print(1UPB[,]INT([]INT(print("there");()))); #1#
  print(2UPB[,]INT(())); #0#
  print(2UPB[,]INT((),(1))) # runtime error, wrong length #
END
```

- - . - -

```
#clau05#
(  #Test vacuum as string #

  PROC p = (STRING s) VOID:
  print((newline, LWB s, UPB s, s));
```

```
  p("") # +1  +0  #;
  p(()) # +1  +0  #;
  p(BEGIN END) #1 0#
)
```

- - . - -

```
#clau06#
BEGIN # Vacuum #
  []INT i=(); print(i[1]) # runtime error, subscript overflow #
END
```

- - . - -

```
#clau07#
BEGIN # If-, case- and ucase-clauses #

  FOR i
  DO print((
        ( i = 1 | 1
        |: i = 2 | 2
        |: i = 3 | 3 | eo if), newline))
  OD;
eo if:

  FOR i
  DO print((
        ( i | 4, 5
        |: i - 2 | 6, 7 | eo case), newline))
  OD;
eo case:

  FOR i
  DO print((
        ( UNION (INT, REAL, CHAR, STRING, BOOL)
         ( i | 1, 1.0, "a", "", TRUE)
        |(INT) : 8, (REAL) : 9
        |: UNION(CHAR, STRING, BOOL) ( i-2 | "a", "", TRUE)
        |(CHAR) : 10, (STRING) : 11
        | eo ucase), newline))
  OD;
eo ucase: SKIP

END
```

- - . - -


```
#clau08#
BEGIN  stop;   # no errors, but loops if not stopped here #

# A:  Statements in the context of a BEGIN block #

BEGIN  label : SKIP;
        11:BEGIN SKIP; SKIP END;

        BEGIN GOTO label; GO TO label END;

        BEGIN INT a1, a2, a3;
        14:FOR i FROM a1BY a2TO a3 DO SKIP OD;
        FOR i FROM a1BY a2TO a3 DO SKIP OD
        END;

        12:BEGIN BOOL a;
            IF a THEN SKIP FI;
            15:IF a THEN SKIP FI
        END;

        BEGIN PROC VOID a; 16: a; a END;

        13 : BEGIN PROC (INT) VOID a; INT b;
            a(b); 17 : a(b)
        END;

        BEGIN REAL a; a:= a; a:= a END;

        BEGIN REF[]REAL a; INT i;
            18 : a[i]:=i; a[i]:=i
        END;

        BEGIN BEGIN SKIP END;
            BEGIN SKIP END
        END
END;

# B:  Statements in the context of a ( block #

(  label : SKIP;
        11:( SKIP; SKIP );

        ( GOTO label; GO TO label );

        ( INT a1, a2, a3;
        14:FOR i FROM a1BY a2TO a3 DO SKIP OD;
        FOR i FROM a1BY a2TO a3 DO SKIP OD
        );

        12:( BOOL a;
            IF a THEN SKIP FI;
```

```
            15:IF a THEN SKIP FI
        );

        ( PROC VOID a; 16: a; a );

        13 : ( PROC (INT) VOID a; INT b;
            a(b); 17 : a(b)
        );

        ( REAL a; a:= a; a:= a );

        ( REF[]REAL a; INT i;
            16 : a[i]:=i; a[i]:=i
        );

        ( ( SKIP );
            ( SKIP )
        )
);

# C:  Statements in the context of IF statement #

BEGIN BOOL true;

    IF true THEN SKIP; SKIP FI;

    IF true THEN SKIP ELSE SKIP; SKIP FI;

    IF true THEN IF true THEN SKIP FI
            FI;

    IF true THEN IF true THEN SKIP FI
            ELSE SKIP FI;

    IF true THEN IF true THEN SKIP ELSE SKIP FI
            FI;

    IF true THEN IF true THEN SKIP ELSE SKIP FI
            ELSE SKIP FI;

    IF true THEN SKIP
            ELSE IF true THEN SKIP FI
            FI;

    IF true THEN SKIP
            ELSE IF true THEN SKIP ELSE SKIP FI;
                IF true THEN SKIP FI; SKIP
            FI

END;
```



```
# D:  Statements in the context of a FOR statement #

BEGIN INT a1, a2, a3; BOOL true;
        PROC VOID proc1; PROC (INT) VOID proc2;
        REAL aa; REF[]REAL bb;

    FOR i FROM a1 BY a2 TO a3
            DO SKIP; SKIP OD;

    FOR i FROM a1 BY a2 TO a3
            DO GOTO stop; GOTO stop OD;

    FOR i FROM a1 BY a2 TO a3
            DO IF true THEN SKIP FI;
            IF true THEN SKIP FI OD;

    FOR i FROM a1 BY a2 TO a3
        DO FOR i FROM a1 BY a2 TO a3
                DO SKIP OD;
            FOR i FROM a1 BY a2 TO a3
                DO SKIP OD OD;

    FOR i FROM a1 BY a2 TO a3
            DO proc1; proc1 OD;

    FOR i FROM a1 BY a2 TO a3
            DO proc2(a1); proc2(a1) OD;

    FOR i FROM a1 BY a2 TO a3
            DO aa := aa; aa := aa OD;

    FOR i FROM a1 BY a2 TO a3
            DO bb [i] := a1; bb[i] := a1 OD;

    FOR i FROM a1 BY a2 TO a3
            DO BEGIN SKIP END;
                BEGIN SKIP END OD

END;

# E:  Statements in the context of a routine declaration #

BEGIN BOOL true; INT a1, a2, a3;
        REF[]INT a4 = a1; REAL a5;
    PROC a = VOID:SKIP;
    PROC b = VOID: IF TRUE THEN SKIP FI;
    PROC c = VOID: FOR i FROM a1 BY a2 TO a3 DO SKIP
                OD;
    PROC d = VOID: d;
    PROC (INT) VOID e = (INT f) VOID: e(f);
    PROC f = VOID: a5:=a5;
```

```
    PROC g = VOID: a4[a1] := a1;
    PROC h = VOID: BEGIN SKIP END;

    SKIP
END
END
```

- - . - -

```
#clau09#
# Optimisation correct ? #
BEGIN

    print((newline,"Prints errors only",newline));

    PROC puti = (INT i, INT p, q) VOID:
    IF p /= q THEN print((i, p, q)) FI;

    PROC putr = (INT i, REAL p, q) VOID:
    IF p /= q THEN print((i, p, q)) FI;

BEGIN
    [1:3]INT a;
    a[1]:=2; a[2]:=3; a[3]:=1;
    putr(1, 44.9104,
        (-1.0+(-2.0+(-3.0+(-4.0+2))))
        +(
        (-5.0+(-2.0-(-5.0+(-2.0-7))))
        -(
        a[a[a[a[2]]]]
        +(
        a[a[a[a[1]]]]
        -(
        a[a[a[a[3]]]]
        + (
        (-1.0*(-2.0*(-3.0*(-4.0*2))))
        -  (
        (-5.0*(-2.0*(-5.0*(-2.0*7))))
        *     (
    128*(-1.0/(-2.0/(-4.0/(-4.0/2))))
        /    (
        (-10.0/(-5.0/(-5.0/(-2.0/2))))
        **
        a[a[a[a[2]]]]
        **
        a[a[a[a[1]]]]
        **
        a[a[a[a[3]]]]
        ))))))))
    )
```



```
END;

BEGIN REAL x; [1:20]REAL a;
    FOR i FROM 1 BY 1 TO 20
    DO a[i] := i-10 OD;
    x := a[1]+(a[2]+(a[3]+(a[4]+(a[5]+(a[6]+(a[7]+(a[8]+(a[9]+
        (a[10]+(a[11]+(a[12]+(a[13]+(a[14]+(a[15]+(a[16]+
        (a[17]+(a[18]+(a[19]+(a[20])))))))))))))))))))-9.0;
    putr(2, 1.0, x)
END;

BEGIN

    [1:10]INT a;

    FOR i FROM 1 BY 1 TO 9 DO a[i]:=i+1 OD;
    a[10]:=1;

    FOR i FROM 1 BY 1 TO 10
    DO puti(3, i, a[a[a[a[a[a[a[a[a[a[a[a[a[a[a[a[a[a[a[a[a[i
            ]]]]]]]]]]]]]]]]]]]]
        )
    OD
END;

    print((newline,"End of tests",newline))

END
```

- - . - -

```
#coer01#
BEGIN # Coercions #
    print ((REAL x:= 0; REF [] REAL (x):= 1; x)); # 1.0 #
    print ((INT n:= 0; n +:= 1:= 5)) # 5 #
END
```

- - . - -

```
#coer02#
BEGIN # Widening #
  FOR i TO 2 DO
  print(
    CASE i IN TRUE,     2r1    OUT []BOOL(TRUE) ESAC
   [CASE i IN  1  , bits_width OUT     SKIP     ESAC]
    ) OD;                                    # TT #

  print(newline);
  FOR n TO 3 DO
```

```
  print((re OF CASE n IN 1, 2.0, 3 I 5 ESAC,
        im OF CASE n IN 1, 2.0, 3 I 5 ESAC))
  OD
                                    # 1.0 0.0, 2.0 0.0, 3.0 5.0 #
END
```

- - . - -

```
#coer03#
BEGIN # Morf versus comorf #

  PROC right = VOID : print("right"),
        wrong = VOID : print("wrong");

  PROC deproc = (STRING mcm) VOID :
    print((newline, mcm, "   deproc: ")),
  PROC nodeproc = (STRING mcm) VOID :
    print((newline, mcm, " nodeproc: "));

  deproc("selection   ");
  proc OF STRUCT(PROC VOID proc, INT d)(right, SKIP);

  deproc("slice       ");
  []PROC VOID(right)[1];

  deproc("routine text");
  PROC VOID : right;

  deproc("formula     ");
  OP + = (INT i) PROC VOID : right; +1;

  deproc("call        ");
  ((INT i) PROC VOID : right)(1);

  deproc("identifier  ");
  right;

  nodeproc("assignation ");
  LOC PROC VOID := wrong;

  nodeproc("cast        ");
  PROC VOID (wrong);

  nodeproc("generator   ");
  LOC PROC VOID;

  FOR i TO 2
  DO IF i = 1
     THEN deproc("balance     "); right
     ELSE nodeproc("balance     "); PROC VOID(wrong)
```



```
     FI
  OD

END
```

- - . - -

```
#coer04#
BEGIN # Coercion error, a unit is not a coercend #
  [] STRUCT (INT i, BOOL j) k = ((1), (TRUE));
  SKIP
END
```

- - . - -

```
#coer05#
BEGIN # Row display cannot be united #
  print(UPB IF FALSE THEN []INT(1) ELSE (1,2,3) FI)
END
```

- - . - -

```
#coer06#
BEGIN # Case clause #
  UNION (INT, REAL) ir,
  UNION (INT, CHAR) ic;
  print(CASE (FALSE|ir|ic) IN
   (INT):1, (REAL):2 ESAC)
  # Error, (p|ir|ic) cannot be meekly balanced #
END
```

- - . - -

```
#coer07#
BEGIN # Weak balance #
    print ((COMPL x:=1;
        CASE 2 IN NIL, IF [] BOOL (TRUE, FALSE)
        [2] THEN REF REF [] COMPL: NIL
            ELSE x FI, LOC PROC REF [] STRUCT (REAL re,im)
            ESAC
        # REF [] COMPL = x # [1]:=3; x))
    #  3.0 I 0.0 #
END
```

- - . - -

```
#coer08#
BEGIN # Soft balance #
    print((HEAP REAL x:= 3.14;
           CASE 3 IN
           SKIP,
           IF x<0 THEN GOTO stop ELSE
           REF [] REAL : NIL FI,
           IF x>0 THEN x ELSE x+:=1 FI
           ESAC:=pi)[1])
    # 3.14159265...#
END
```

- - . - -

```
#coer09#
BEGIN # Soft balance #
    print (CASE 2 IN SKIP, NIL,
        IF BOOL (SKIP) THEN GOTO stop ELSE
        PROC REF [] INT (SKIP) FI ESAC:=:
        CASE 3 IN LOC REF REF [] INT, LOC INT, NIL
    ESAC)
    #TRUE, would you believe #
END
```

- - . - -

```
#coer10#
BEGIN # Union with VOID #

  OP TOPROCINT = (INT i) PROC INT : INT : 1;
  OP TOVOID = (INT i) VOID : 1;

  STRING proc int = "proc int", void = "void",
    before = "before ", after = " after";

  print(("Results must be:", newline,
        void, newline,
        proc int, after, 1, newline,
        before, void, newline,
        before, void, newline,
        before, void, newline,
        proc int, 1, newline,
        void, newline,
        proc int, 1, newline,
        proc int, after, 1, newline,
        before, void, newline,
         newline, "Results are:", newline));

  UNION(PROC INT, VOID) upiv := EMPTY;
```



```
  PROC pupiv = VOID:
    print((upiv
        |(PROC INT pi) : ((print(proc int); pi), newline)
        |(void, newline)));

  pupiv;
  upiv:= INT : (print(after); 1);
  pupiv;

  upiv:= VOID : (print(before); 1);
  pupiv;

  upiv:= VOID ((print(before); 1));
  pupiv;

    # firm void position #
  upiv:= print(before);
  pupiv;

  upiv:= TOPROCINT 1;
  pupiv;

  upiv:= TOVOID 1;
  pupiv;

  upiv:= INT : 1;
  upiv:= label # must jump before assigning #; print("Error 1");
label:
  pupiv;

  FOR i TO 2
  DO upiv:=
    CASE i IN
         INT : (print(after); 1),
         VOID : (print(before); 1)
    ESAC;
    pupiv
  OD

END
```

- - . - -

```
#coer11#
BEGIN # Contains all possible two-member coercion sequences #

  UNION(INT, BOOL) ib:= 1;

  print(([]REAL(1), newline));
  print(([]REAL(INT : 1), newline));
```

```
  print(([]REAL(REAL : 1), newline));

  print(([] [,]COMPL (1), newline));
  print(([] [,]COMPL (LOC INT:= 1), newline));
  print(([] [,]COMPL ([]COMPL(1, 2)), newline));

  print(([,,] [] BOOL (16 r f), newline));
  print(([,,] [] [,] BOOL (16 r f), newline));
  print(([,,] [] BOOL (BITS : 16 r f), newline));
  print(([,,] [] [,] BOOL (BITS : 16 r f), newline));

  print(([,] [] CHAR (bytes pack ("ab")), newline));
  print(([,] [] [,] CHAR (bytes pack ("ab")), newline));
  print(([,] [] CHAR (LOC BYTES:= bytes pack("ab")), newline));
  print(([,] [] [,] CHAR(LOC BYTES:= bytes pack("ab")), newline));

  print((REF[]INT(REF INT : HEAP INT:= 1), newline));
  print((REF[,]INT(REF[]INT : HEAP[1]INT:= 1), newline));

  print((UNION(INT, REAL, BOOL) (ib), newline));
  print(( [] REF [] [,] [] [] INT
          (LOC PROC REF INT:= REF INT : HEAP INT:= 1) [1]
          , newline));

  print(([]UNION(INT, REAL) (LOC INT:= 1) [1], newline));
  print(([]UNION(INT, REAL) (REAL : 1) [1], newline));
  print(([]UNION(INT, REAL, BOOL) (ib) [1], newline));

SKIP END
```

- - . - -

```
#coer12#
BEGIN # Bad unions with VOID #

  UNION(REAL, VOID)( 1.0, 2.0);
  UNION(REAL, VOID) PAR ( 1.0, 2.0);
  UNION(REAL, VOID) (DO SKIP OD);
  UNION(REAL, VOID)  DO SKIP OD

END
```

- - . - -

```
#coer13#
BEGIN # Soft balance with EXIT's #

  INT i; [ 1 : 1 ]INT ri, rj;
  PROC pri = REF[]INT : rj;
```



```
  FOR c TO 3
  DO
    ([]PROC VOID switch = ( lrri, li, lpri);
      switch[c]; SKIP                          # hip #
    EXIT lrri: LOC REF[]INT := ri              # deref #
    EXIT li: i                                 # row #
    EXIT lpri: pri                             # deproc #
    ) := c
  OD;

  print((ri, i, pri, newline))                 # 1 2 3 #
END
```

- - . - -

```
#coer14#
( # Rowing of NIL yields NIL #
    print(("print: ", TRUE, " ",
              REF[]INT(NIL) :=: REF INT(NIL), newline)))
```

- - . - -

```
#idr101#
BEGIN # Identity relations #
  REAL x; REF REAL y:= x;
  print((x:=:y, y:=:x, newline)) # TRUE, TRUE #;
  print((x:=: REF[]REAL(x)[1], newline)) # TRUE #;
  print((x:=: REF[]REAL(x)   , newline)) # FALSE #
END
```

- - . - -

```
#idr102#
BEGIN
    REAL a;
    a :=: (1);              # correct, 1 = REF REAL #
    a :=: 1;                # incorrect, 1 is a unit, not a tert
    1: SKIP;

    IF INT i, j, k, l; i:=:j AND k:=:l # illegal formula #
    THEN SKIP FI
END
```

- - . - -

```
#stow01#
BEGIN

  print(("Results must be:", newline,
          FALSE, TRUE, TRUE, FALSE, newline,
          1, 1, TRUE, TRUE, newline,
          2, 2, FALSE, FALSE, newline,
          newline,
          1, newline,
          2, 1, 2, newline,
          3, 2, 3, newline,
          []COMPL((0, 0), (1, 1), (0, 0)), newline,

          newline, "Results are:", newline));

  [1:2] PROC BOOL i; INT j;
  i[1]:= BOOL: j=2; i[2]:= BOOL : j=1;
  j:= 1; print(i[1]); print(i[2]);
  j:= 2; print(i[1]); print(i[2]);

  print(newline);
  [] STRUCT(INT i, BOOL j) k =((1, TRUE), (2, FALSE));
  FOR i TO UPB k
  DO
    print(((i OF k)[i], i OF k[i], (j OF k)[i], j OF k[i],
            newline))
  OD;

  print(newline);
  print( a OF (STRUCT(INT a, b) s = (1, 0); s));

  print(newline);
  [ 2 : 3 ] INT cc;
  print((LWB cc, LWB cc[:], LWB cc[]));
  print(newline);
  print((UPB cc, UPB cc[:], UPB cc[]));

  print(newline);
  [ 1 : 3 ] COMPL r:= (0, (0, 1), 1);
  re OF r:= im OF r; print(r);

  SKIP
END
```

- - . - -

```
#stow02#
BEGIN # Some slices #
  [0:7][0:15] INT a;
  INT n:= 0;
```



```
  FOR i TO 8 DO FOR j TO 16 DO a[i-1][j-1]:= n+:=10D OD;
  print(a[0][15]); #16#
  print(a[0:0 AT 0][0][15]);     #16#
  print(a[0:0 AT 0][0:0 AT 0][0][5:15][11:13 AT 2][4]); #16#
  print(newline);

  []INT k = a[0:0][15] # wrong, a[0:0] has bounds [1:1][0:15],
          so there occurs overflow #;
  SKIP
END
```

- - . - -

```
#stow03#
BEGIN [1:-1] INT k; print("OK"); k[1]:= 1 # overflow # END
```

- - . - -

```
#stow04#
BEGIN
  print([][] BOOL(TRUE, 2r1)) ; # TF...FT #
  print(newline);
  print([,] BOOL(TRUE, 2r1)) # runtime error, wrong length #
END
```

- - . - -

```
#stow05#
(     ""[];   # OK #
      "a"[];  # KO #
      "ab"[]  # OK #
)
```

- - . - -

```
#stow06#
BEGIN # Against over-optimization of string comparison #

    STRING str = "string with step > 1"; STRING ref str:= str;
    [1:UPB str] STRUCT(REAL flub, CHAR c) rst;

    print(("Result must be:",
        newline, str, ".", newline, str, ".", newline,
        "First test OK", newline, "Second test OK", newline,
        newline, "Result is:", newline));

    FOR i TO UPB str DO c OF rst[i]:= ref str[i] OD;
```

```
    print((c OF rst, ".", newline, ref str, ".", newline));

    IF c OF rst = str AND c OF rst = ref str
    THEN print(("First test OK", newline))
    ELSE print(("Erroneous string, is: ", c OF rst,
                            " , must be: ", str, newline))
    FI;

    c OF rst:= str;
    IF c OF rst /= str OR c OF rst /= ref str
    THEN print(("Erroneous string, is: ", c OF rst,
                            " , must be: ", str, newline))
    ELSE print(("Second test OK", newline))
    FI
END
```

- - . - -

```
#stow07#
BEGIN # Test  +  and  =  on strings #

    PROC equal = (STRING a, b) BOOL:
    (   INT p = UPB a - LWB a + 1,
            q = UPB b - LWB b + 1;
        INT r = ( p > 0 | p | 0 ),
            s = ( q > 0 | q | 0 );

        IF r /= s THEN FALSE
        ELSE BOOL c:= TRUE;
            INT la = LWB a - 1, lb = LWB b - 1;
            FOR i TO r WHILE c:= a[i+la]=b[i+lb] DO SKIP OD;
            c
        FI
    );

    PROC concat = (STRING a, b) STRING:
    (   INT p = UPB a - LWB a + 1,
            q = UPB b - LWB b + 1;
        INT r = ( p > 0 | p | 0 ),
            s = ( q > 0 | q | 0 );
        [ r + s ] CHAR c;

        (   c[   1 :   p @ LWB a ]:= a,
            c[ r+1 : r+q @ LWB b ]:= b);
        c
    );

    print(("This program should print a 25 * 25 block of stars.",
            newline, newline));
    FOR lwb a FROM -2 TO 2 DO
```



```
        FOR sze a FROM -2 TO 2 DO
            print(newline);
            FOR lwb b FROM -2 TO 2 DO
            FOR sze b FROM -2 TO 2 DO
                STRING a = "1a" [ 1 : sze a @ lwb a ],
                         b = "1b" [ 1 : sze b @ lwb b ];

                print("*"); # to estimate progress #

                # test = #
                IF ( a=b ) = equal(a, b) THEN SKIP
                ELSE print((newline, "Error in string comparison: ",
                        "should be ", ( equal(a, b) | "" | "un"), "equa
                        ", are ", ( a=b | "" | "un"), "equal"));
                    GOTO bad
                FI;

                # test + #
                IF a + b = concat(a, b) THEN SKIP
                ELSE print((newline, "Error in string concatenation: ",
                        "should be """, concat(a, b),
                        """, is """, a+b, """"));
                    GOTO bad
                FI
            EXIT bad:
                print((newline,
                    "  first string is """, a,
                    """, lwb=", whole(LWB a, 4),
                    ", upb =", whole(UPB a, 4),
                    ", second string is """, b,
                    """, lwb=", whole(LWB b, 4),
                    ", upb =", whole(UPB b, 4),
                    newline))
            OD OD
        OD OD

END
```

- - . - -

```
#stow08#
BEGIN  # Test king size indices and midget slices #
    INT i := 0;
    WHILE i <= (maxint-1) OVER 2
    DO
        INT maxdex = i:= 2*i+1; INT mindex = -maxdex;
        print((newline, "Bounds:          ", maxdex, mindex, newli
        [maxdex : maxdex] REAL maxvec; maxvec[maxdex]:= 1;
        print(("Bounds of maxvec: ",
                LWB maxvec, UPB maxvec, newline));
```

```
        [mindex : mindex] REAL minvec; minvec[mindex]:= 1;
        print(("Bounds of minvec: ",
                LWB minvec, UPB minvec, newline));

        HEAP [maxdex : mindex] REAL flatvec;
        print(("Bounds of flatvec:",
                LWB flatvec, UPB flatvec, newline));

        REF [] REAL u := flatvec; HEAP [1] REAL v;
        v[LWB u:UPB u @ LWB u]:= u
    OD
END
```

- - . - -

```
#flex01#
( # OK #
    MODE S = FLEX [1 : 0] CHAR, T = [1 : 0] CHAR;
    REF STRING n = LOC S:= "Next line will be empty, then a";
    UNION(REF S, REF T) f = LOC T := "";
    UNION(STRING, CHAR) u = UNION(S, T, CHAR) ("a");

    print((n, newline));
    print(((f | (REF S s):s , (REF T t):t), newline));
    print((u, newline))
)
```

- - . - -

```
#flex02#
BEGIN # Transiency tests, all OK #

  BOOL b = TRUE, y = FALSE;

  print((
   IF b THEN LOC CHAR ELSE (LOC STRING)[1] FI:= "a",
        newline));

  print((
   IF b THEN LOC[1:3]CHAR ELSE (LOC STRING)[] FI:= "bcd",
        newline));

  print((
   IF y THEN LOC STRING ELSE LOC[1:1,1:3]CHAR FI:= "efg",
        newline))
                    #  a
                       bcd
                       efg  #
END
```


- - . - -

```
#flex03#
BEGIN # All erroneous #
  MODE STRENG = FLEX [1:1] CHAR;

  LOC REF CHAR:= (LOC STRENG)[1];
    # nontrans #            # trans #

  LOC REF CHAR := (TRUE | LOC CHAR | (LOC STRENG)[1]);
    # nontrans #                  # nontrans #        # trans #

  (TRUE | LOC[1:3]CHAR | LOC STRENG):= "abc";
             # nonflex #          # flex #

  (LOC STRENG)[] :=: (LOC STRENG)[]
     # trans #              # trans #
END
```

- - . - -

```
#flex04#
(   print(("Ghost element test. Results should be:", newline,
      3, "  3", newline, "followed by error in multiple assignation",
      newline, newline, "Results are:", newline));

    FOR k FROM 3 TO 4
    DO FLEX [1:0] [1:3] CHAR flex fix;
        flex fix:= "  34"[ : k];
        print(k); print(flex fix); print(newline)
    OD
)
```

- - . - -

```
#garb01#
BEGIN # Test garbage collector #
  TO 1000 DO HEAP [ 1000 ] REAL OD;
  print(("collections, garbage, collect seconds:", newline));
  print(( collections, garbage, collect seconds  , newline))
END
```

- - . - -

```
#garb02#
BEGIN # Test garbage collector #
  REF [] REAL x, y, INT n:=0;
```

```
1: x:= HEAP[1:1000] REAL; y:= HEAP[1:1000] REAL;
  IF (n+:=1)<1000 THEN 1 FI;
  print(("collections, garbage, collect seconds:", newline));
  print(( collections, garbage, collect seconds  , newline))
END
```

- - . - -

```
#garb03#
BEGIN # Test garbage collector #

  INT size = 250;

  REF [] REAL x, [1: size ] REF [] REAL y, INT n:= 1;
1: x:= HEAP[1: n ] REAL;            # to throw away #
  y[n]:= HEAP[1:10] REAL;           # to be kept #
  FOR k TO 10 DO y[n][k]:= 10*n + k - 11 OD;

  FOR m TO n DO
    FOR k TO 10 DO
      IF y[m][k] /= 10*m + k - 11
      THEN print((newline, "Error in element", m, k,
                  "value is", y[m][k], " should be", 10*m + k - 11,
                  new line,
                  "after", collections, " garbage collections"))
      FI
    OD
  OD;

  IF (n+:=1) LE size THEN 1 FI;

  print(("collections, garbage, collect seconds:", newline));
  print(( collections, garbage, collect seconds  , newline))

END
```

- - . - -

```
#garb04#
BEGIN # Heap #
  INT n:= 0, REF INT x:= LOC INT :=0;
 1: (HEAP INT p:= n+:= 1; print((x, p)); # 0,1,1,2,2,... #
  x:= p); (n<100|1);
  print(newline);
  print(("collections, garbage, collect seconds:", newline));
  print(( collections, garbage, collect seconds  , newline))
END
```

- - . - -


```
#scop01#
BEGIN # Scope error #
  print(("Need not run", newline));

  PROC pp = (INT i) PROC INT: INT : i + 1 # error #;
  print(pp(1))

END
```

- - . - -

```
#scop02#
BEGIN #Scope error#
  print(("Need not run", newline));
    PROC VOID pv= (1: VOID:
        (MODE M1= [1:($n((1; HEAP INT):= 3) "a" $; 1)] INT;
        M1 x:= 1; SKIP ));
    pv
END
```

- - . - -

```
#scop03#
BEGIN # No scope error #
    PROC VOID pv= (1: VOID:
        (MODE M1= [1:($n(( #1;# HEAP INT):= 3) "a" $; 1)] INT;
        M1 x:= 1; SKIP ));
    pv;

    print("End of test")
END
```

- - . - -

```
#scop04#
  # Routine scope error #
BEGIN
  print(("Need not run", newline));


MODE FUN = PROC(INT)INT;
MODE OPERATOR = PROC(FUN)FUN ;
OPERATOR nabla = (FUN t)FUN :
                    (INT x)INT : t(x)-t(x-1);
OP UP = (OPERATOR a, INT b)OPERATOR :
            (FUN f)FUN : (b=0 | f | a((a UP (b-1))(f)));
PRIO MIN = 1;
OP MIN = (INT a,b)INT : (a<=b | a | b);
```

```
FUN pol4 = (INT x)INT : x*(x+1)*(x+2)*(x+3);


FOR n FROM 0 TO 20
DO
   print(n);
   FOR k FROM 0 TO (n-1) MIN 5
   DO
      print((nabla UP k)(pol4)(n))
   OD;
   print(newline)
OD


END
```

- - . - -

```
#scop05#
  # No scope error #
( [ 8 ] REF [] INT a;

  FOR i TO UPB a  # non-local #
  WHILE PRIO + = 3; TRUE  # non-local #
  DO  # non-local #
    CASE UNION(INT, REAL)(i)  # non-local #
    IN (INT k) :  # non-local #
      BEGIN l: # non-local #
        a[k]:= LOC [k] INT;
        FOR i TO k DO a[k][i]:= k + i OD
      END
    ESAC
  OD;

  print(("A triangle of integers, ascending downwards and to the right",
          newline));
  FOR i TO UPB a
  DO print((a[i], newline)) OD
)
```

- - . - -

```
#scop06#
BEGIN # Scope error #
  print(("Need not run", newline));
  REF INT ii;
  MODE A = [ii:= LOC INT:= 1]BOOL;
  LOC A; print(ii)
END
```


- - . - -

```
#scop07#
BEGIN # Scope error, parameter pack is local #
  print(("Need not run", newline));

  print((LOC INT:= 0) +:= 1);
  REF INT ii;
  print(sin(ii:= LOC INT:= 1));
  print(ii)
END
```

- - . - -

```
#scop08#
BEGIN # Scope error #
  BEGIN (REF REF INT p) VOID:
              (p:= LOC INT:= 3; print(p))
  END (LOC REF INT)
END
```

- - . - -

```
#scop09#
BEGIN # Scope error #
  print(("Need not run", newline));

  PROC VOID pv =
  (MODE M = INT;
    VOID: HEAP M # actual declarer #
  );

  PROC VOID qw =
  (MODE M = [1:a] INT; INT a = 1;
    VOID: HEAP M # actual declarer #
  );

  pv;
  qw

END
```

- - . - -

```
#scop10#
BEGIN # No scope error #
```

```
  PROC VOID pv =
  (MODE M = INT;
    VOID: HEAP REF M # formal declarer #
  );

  PROC VOID qw =
  (MODE M = [1:a] INT; INT a = 1;
    VOID: HEAP REF M # formal declarer #
  );

  pv;
  qw

END
```

- - . - -

```
#jump01#
  # Simple jumps and EXIT's #
( FOR i TO 2 DO
      IF i = 2 THEN GOTO l FI; print("First") EXIT
    l:print(" second")
  # Result:  First second #OD )
```

- - . - -

```
#jump02#
BEGIN REAL a; GOTO l; INT i:= 1; l: print(i)
  # the declaration of 'i' has not been elaborated #
END
```

- - . - -

```
#jump03#

BEGIN # Jump #
  INT i:=1, j:=2;
  i:= j:= (GOTO l; 3); l: print((i, j)) # 1  2 #
END
```

- - . - -

```
#jump04#
( # Jump out of procedure #

 ( # directly #
  PROC jump = VOID: ( print(2); print((9, GOTO l, 8)) );
```



```
   print(1); jump; print(7);
l: print(3)
 );

 ( # indirectly #
  MODE HIDE = PROC VOID;
  HIDE p = ( TRUE | GOTO m);
  print(4); p; print(6); m: print(5)
 )
)                              # result is 1, 2, 3, 4, 5 #
```

- - . - -

```
#jump05#
BEGIN # Test stack jump in ALGOL 68, Dick Grune, 24-07-73.
    A bit-pattern is decomposed on the stack into a sequence of PROC
    VOID's, the bit-pattern is re-assembled by calling the deepest
    PROC VOID and the resulting pattern is compared to the original.
    #
  INT max width = 12;

# Additional BITS-operators #

    INT conv = bits width - max width;

    BITS one = BIN 1 SHL (bits width - 1);

    OP SET = (INT i, REF BITS rb) REF BITS:
        rb:= rb OR one SHR (i - 1);

    OP NEXT = (BITS b) BITS:
    BIN (ABS (b SHR conv) + 1) SHL conv;

    PRIO SET = 9;

    BITS max bits = BIN(2 ** max width - 1) SHL conv;

# End #

    PROC dive = (INT level, PROC VOID back) VOID:
    (   IF level > max width THEN back ELSE
            dive(level + 1,
                IF level ELEM bits THEN here ELSE back FI)
        FI;
    here: level SET acc; back
    ) # dive #;

# Try all (4096) bit-patterns #

    BITS bits # proposed pattern # := BIN 0,
```

```
        acc  # assembled pattern #;

    INT cnt:= 0;
    WHILE
      acc:= BIN 0; (dive(1,out); out:SKIP# it just happened #);
      IF bits NE acc
      THEN print(("Stack jump test failed. Bits: ", bits,
            "   acc: ", acc, newline)); stop
      FI;
      bits NE max bits # WHILE #
    DO bits:= NEXT bits; cnt +:= 1 OD;
    IF cnt /= 2 ** max width - 1 THEN
       print(("Something wrong", cnt, 2 ** max width - 1)); stop
    FI;
    print(("Stack jump test successful", newline))
END
```

- - . - -

```
#par101#
BEGIN

  # Co-routines simulated by parallel processing.
    'Invert' is a routine that accepts a stream of characters, inverts
    all letter-sequences (words) in it and yields the resulting stream
    of characters. It cooperates in a co-routine fashion with a second
    call of itself so that the net result is the original stream of
    characters. Process 1 reads from the 'reader' and writes on the
    interface, process 2 reads from the interface and writes on the
    printer. The program causes extensive process switching.
  #

  # Reader #
  PROC read = (REF CHAR res) VOID:
    res:= next(rp, "this is a readable text  withalongwordinit.");
  PROC next = (REF INT p, STRING st) CHAR:
    IF p >= UPB st THEN end of file ELSE st[p +:= 1] FI;
  CHAR end of file = REPR 128;  INT rp := 0;
  # End of reader #

  PROC invert = (INT proc) VOID:
  WHILE
    CHAR term = word(proc); out(term, proc); term /= end of file
  DO SKIP OD;

  PROC word   = (INT proc) CHAR:
    # inverts the word (which may be empty) and yields its terminator #
  ( CHAR s; in(s, proc);
      IF letter(s) THEN
        CHAR t = word(proc); out(s, proc); t # invert #
```



```
        ELSE s FI
    ) # word # ;

    PROC letter = (CHAR c) BOOL: "a" <= c AND c <= "z";

    PROC in = (REF CHAR res, INT proc) VOID:
    IF proc = first THEN read(res) ELSE
      DOWN read OF interface;
      res:= item OF interface;
      UP write OF interface
    FI # in # ;

    PROC out = (CHAR res, INT proc) VOID:
    IF proc = last THEN print(res) ELSE
      DOWN write OF interface;
      item OF interface:= res;
      UP read OF interface
    FI # out # ;

    STRUCT (SEMA write, REF CHAR item, SEMA read) interface =
           (LEVEL 1    , LOC CHAR     , LEVEL 0  );

  # Program # INT first = 1, last = 2;
    PAR ( invert(first), invert(last))
  END
```

- - . - -

```
  #par102#
  BEGIN # Parallel sorting #

      PROC sort = (REF [] INFO a) VOID:
      IF INT n items = UPB a; n items > 1
      THEN # A row of (n items - 1) parallel sorters is
             constructed. They run until they are all satisfied.
             this is tested by keeping a count of the number
             of unsatisfied sorters.
           #

          [] SEMA guard a = # boolean sema's for items in 'a' #
             ([n items] SEMA s;
              FOR i TO n items
              DO s [i]:= LEVEL 1 # available # OD;
              s);

          INT n sorters = n items - 1;
          [] SEMA sorter =
             ([n sorters] SEMA s;
              FOR i TO n sorters
              DO s[i]:= LEVEL 1 # active # OD;
```

```
        s),

    SEMA guard nus = LEVEL 1,
    INT nus # number of unsatisfied sorters # := n sorters,
    SEMA finished = LEVEL 0 # completion bit #;

    PROC build sorters = (INT n) VOID:
    PAR BEGIN
        DO start sorter(n) OD,
        IF n>1 THEN buildsorters (n-1) FI
    END;

    PROC start sorter = (INT n) VOID:
    (DOWN sorter[n];
        IF (DOWN guard a[n], DOWN guard a[n+1]);
            BOOL exch = a[n+1] < a[n];
            IF exch THEN INFO p = a[n+1];
                a[n+1]:= a[n]; a[n]:= p
            FI;
            (UP guard a[n], UP guard a[n+1]);
            exch
        THEN IF n > 1 THEN wake(n-1) FI;
            IF n < n sorters THEN wake(n+1) FI
        FI;
        stop(n)
    ) # start sorter #,

    PROC wake = (INT n) VOID:
    (DOWN guard nus; nus +:= 1;
    UP sorter[n]; UP guard nus),

    PROC stop = (INT n) VOID:
    (DOWN guard nus; nus -:= 1;
    IF nus = 0 THEN UP finished FI; UP guard nus);

  ##
    PAR BEGIN
        # someone looking at the completion bit #
            (DOWN finished; GOTO 1),
        # the sorters # build sorters (n sorters)
    END; 1: SKIP
  FI # sort #;

MODE INFO = INT;

PROC  shuffle= (REF [] INT a) VOID:
BEGIN INT p= LWB a, q= UPB a;
  FOR i FROM q BY -1 TO p+1
  DO REF INT t= a[ENTIER (random * (i-p+1)) + p], u = a[i];
    INT h= t; t:= u; u:= h # swap #
  OD
```



```
  END # shuffle #;

  INT max=  8; [ max ] INT p;

  PROC  test= (PROC (INT) INT a) VOID:
  ( FOR i TO max DO p[i]:= a(i)OD;
    shuffle(p); print(("Shuffled:",newline,p,newline));
    sort(p); print(("Sorted:",newline,p,newline,newline))
  );

  test((INT p) INT: p);
  test((INT p) INT: ENTIER (p/5));
  test((INT p) INT: 0)
END
```

- - . - -

```
#par103#
( # Simple deadlock # SEMA s = LEVEL 0;
      VOID PAR(DOWN s, DOWNs); print("Escaped")
)
```

- - . - -

```
#par104#
( # Uninitialized sema, will it wreck the run-time system ? #
      PAR(DOWN LOC SEMA, DOWN LOC SEMA); print("Escaped")
)
```

- - . - -

```
#par105#
( # Action on sema outside parallel-clause #
  SEMA s = LEVEL 1; DOWN s; print(("Escaped once", newline));
  DOWN s; print(("Escaped twice", newline))
)
```

- - . - -

```
#par106#
( # Sema with negative initial value #
  SEMA ten = LEVEL -10;
  PAR((DOWN ten; print(" second")),
    (TO 10 DO UP ten OD; print("First"); UP ten)
  )
)
```

- - . - -

```
#smio01#
formatless
transput:
BEGIN   #Formatless tests-
              create a file
              write on the file
              read the file
              The reading of the file should produce the same info as
                 was written#

        FILE ti,to;
        #Use a channel with bi-directional properties#
        establish(ti, "ti", stand back channel, 10, 60, 136);

        to:= ti; # 'to' is now open; use it#

        #try something#
        [1:100] INT rj;
        INT j:= 505; REAL x:= 3.14159; COMPL c:= (2.01, 3.10);
        BOOL t:= TRUE;
        FOR i TO UPB rjDO rj[i]:=iOD;
        put (to, (newpage, newline, j,x,c,t,rj));

        #try characters#
        put(to, ("*" # no preceding space#, newline, "*" #no
                  preceding space again#));
        put(to, (newline, "*", backspace, "x" #overwrite the *#));

        #try string#
        STRING s:= "i am a string",
                 s2:= "me too";
        put(to, (newline, s));
        TO UPB s DO backspace(to)OD;
        put(to, s2);
        backspace(to);
        put(to,s2); #write over last character#
              #  yields  "me tome toong"   #

                        #now let's check the file#
        reset(ti);# we have filled "to" and shall read from "ti" #
        [1:UPB rj] INT rj2;
        INT j2; REAL x2; COMPL c2; BOOL t2; STRING u,u2;
        get(ti,(newpage,newline,j2,x2,c2,t2,rj2));
        FOR i TO UPB rj
        DO (rj[i] /= rj2[i]
             | print(("Error1", rj[i] - rj2[i], newline)))
        OD;
```



```
        IF j/=j2 OR x/=x2 OR c/=c2 OR t/= t2 THEN
          print(("Error2", j, j2, x, x2, c, c2, t, t2, x-x2, c-c2, t=t2,
               newline))
        FI;

        CHAR char1, char2;
        get(ti,(char1,newline,char2));
        IF char1/= "*" OR char2/= "*" THEN
          print(("Error3", char 1, char 2, newline))
        FI;

        get(ti,(newline, char1, backspace, char2));
        IF char1 /= char2 OR char2 /= "x" THEN
          print(("Error4", char 1, char 2, newline))
        FI;

        [] CHAR char5 = ("m","e"," ","t","o",
              "m", "e", " ", "t", "o", "o", "n", "g");
        [1:UPB char5] CHAR char6;
        get(ti,(newline, char6 #at end of file#));
        FOR i TO UPB char 5
        DO (char5[i] /= char6[i]
             | print(("Error5", ABS char 5[i], ABS char 6[i],
                      newline)))
        OD;

        #test EOF-stuff#
        on logical file end(ti,(REF FILE f)BOOL:okay);
        get(ti, char1); #should cause call to 'logical
                                  file end' to be generated #
        #if we continue here, then there was an error#
        print(("Error6", newline));

        okay:
        print(("End of test",newline))
END
```

- - . - -

```
#smio02#
BEGIN

  print(("Results must be:", newline,
         3, newline, 3.0, newline, 3, 3.0, newline, 2r11, TRUE,
         newline, 2, 0, newline, 1, newline, TRUE, newline, 6.0,
         newline, "Empty", newline,
         "Correct jump out of print parameter",
         newline, newline, "Results are:", newline));

  print(IF TRUE THEN 3 ELSE 3.0 FI);
```

```
  print(newline);
  print(IF FALSE THEN 3 ELSE 3.0 FI);
  print(newline);
  print(IF TRUE THEN (3, 3.0) ELSE
                    (2r11, TRUE, newline) FI);
  print(newline);
  print(IF FALSE THEN (3, 3.0) ELSE
                    (2r11, TRUE, newline) FI);
  print((INT i:= 1; (i + 1, i - 1, newline) # coll. clause #
              # serial clause # ));

  print(UNION([] INT, BOOL) ([] INT (1)));
  print(newline);
  print(UNION([] INT, BOOL) (BOOL (TRUE)));
  print(newline);
  # 'print' works on a union of everything, so also on BOOL or []INT #

  print((PROC PROC REF REAL: PROC REF REAL:
               REF REAL: HEAP REAL) := 6);
  print(newline);

  print(()); print("Empty");
  print(newline);

  print((3, sqrt( GOTO 1 ),5));  print("Error");
1:print("Correct jump out of print parameter")
END
```

- - . - -

```
#smio03#
BEGIN
# 10/08/73, R van Vliet; 30/09/75, revised.
Test the print and putroutines.#

    INT max ch n = # actual max char, formerly
        max char[standout channel] #
    (FILE f:= standout; INT i;
        on line end(f, (REF FILE f)BOOL: out);
        DO put(f, space) OD;
    out: i:= char number(f) -1; TO i DO put(f, backspace) OD; i);

    print(("Test 1", newline));
    print(("Test rather easy output", new line));
    COMPL z= -max real I -max real;
    print((-max int, -max real, z, FALSE, "a", newline));
    print(new line);
    MODE LINTREAL =UNION(
      INT, LONG INT, LONG LONG INT,
      REAL, LONG REAL, LONG LONG REAL
```



```
      );
    PROC maxim =(LINTREAL lir)LINTREAL:
      CASE lir IN (INT): max int,
        (LONG INT): long max int -LENG 1,
        (LONG LONG INT): long long max int -LENG LENG 2,
        (REAL): max real,
        (LONG REAL): long max real -LENG 1.0,
        (LONG LONG REAL): long long max real -LENG LENG 2.0
      ESAC;
    PROC lengthen =(LINTREAL lir)LINTREAL:
      CASE lir IN
        (INT k): LENG k,
        (LONG INT k): LENG k,
        (LONG LONG INT k):
          (print((new line, "No more long ints allowed")); k),
        (REAL k): LENG k,
        (LONG REAL k): LENG k,
        (LONG LONG REAL k):
          (print((newline, "No more long reals allowed")); k)
      OUT print((new line, "Lengthen called with illegal mode.",
         new line)); GOTO stop
      ESAC;

    LINTREAL lir := max int; print(lir);
    TO int lengths -1
    DO lir :=maxim(lengthen(lir)); print(lir) OD;
    LINTREAL int max =lir;
    lir:=lengthen(lir);
    print((new line, "The result of trying an extra long int is:",
                    lir,newline,newline));

    lir :=max real; print(lir);
    TO real lengths -1
    DO lir:= maxim(lengthen(lir)); print(lir) OD;
    lir :=lengthen(lir);
    print((newline, "The result of trying an extra long real is:",
                 lir, newline));
    print(new line);

    INT digitcount =
      # count the digits in int max #
      CASE int max IN
        (INT) : int width,
        (LONG INT): long int width,
        (LONG LONG INT) : long long int width
      OUT print((newline, "The actual mode of intmax is wrong",
        newline));GOTO stop
      ESAC;
    print(newline);
    TO max ch n -(digitcount +2) DO print(space) OD;
    print(int max);
```

```
    print((new line,
          "This integer must be printed at the end of a line",
      new line));
    TO max ch n -(digitcount +2) +1 DO print(space) OD;
    print(int max);
    print((new line,
        "and this integer at the beginning of a line",
      new line));

    TO max ch n -(2*(real width +exp width) +11)
    DO print(space) OD;
    print(z);
    print((new line,
      "This compl must be printed at the end of a line", new line));
    TO max ch n -(2*(real width +exp width) +11) +1
    DO print(space) OD;
    print(z);
    print((new line,
      "and this last compl at the beginning of a line", new line));

    print((newline, "Three times pi, in stepwise receding positions:",
           newline));
    print((pi, newline));   # no space #
    print((" ", float(pi, real width +exp width +4, real width -1,
                          exp width +1), newline));  # one space #
    print((" ", pi, newline));  # two spaces #

    TO max ch n -4 DO print (space) OD;
    print("lineoverflow");
    print((new line, "[]CHAR was tested", new line, new line));
    print(("Finally print a false and a true boolean", newline,
      FALSE, TRUE))
END
```

- - . - -

```
#smio04#
BEGIN
# 10/08/73, R van Vliet; 30/09/75, revised.
Test the print and putroutines.#

    INT max ch n = # actual max char, formerly
        max char[standout channel] #
    (FILE f:= standout; INT i;
        on line end(f, (REF FILE f)BOOL: GOTO out);
        DO put(f, space) OD;
    out: i:= char number(f) -1; TO i DO put(f, backspace) OD; i);

  print(("Test 2", new line,
    "Test layout-procedures", new line, newline));
```



```
    print((new line, "Check space, backspace and character number",
                newline));
    BEGIN
      INT inspect, k;
      PROC ilchcount =VOID:
        (INT i =char number(standout);
       print((newline,
         "Illegal character number", i, "at position", k, newline));
       GOTO printdots
        );
      BOOL line end;
      FILE auxout:=standout;
      on line end (    auxout ,(REF FILE f)BOOL:
        (  inspect :=char number(standout);
          print(backspace);
          line end := TRUE)
        );
      BEGIN
        k:= 1; line end := FALSE #'on line end' not called yet#;
        WHILE NOT line end DO
           IF char number (standout) NE k
          #Check the character count. Be aware that 'auxout'
          and 'standout' refer to the same book.#
          THEN ilchcount
          ELSE k +:=1; put(auxout, space)
        FI #end of line reached# OD;
        IF max ch n /= inspect -1 THEN
            print((newline,
               "Not all lines of standout have the same length",
                newline))
         FI;
        k -:= 1;
        line end := FALSE;
        TO max ch n DO
        IF k NE char number(standout)
          THEN ilchcount
          ELSE k -:=1; put(auxout, backspace)
        FI # back at the beginning of the line# OD;
        print((new line,
                "This line should be preceded by one blank line"));
        print((new line,
          "Char number of standout is at most",
                max ch n, new line))
      END;

    printdots:
      print((new line,
        "Print 3 lines, having a dot at every second position",
       newline));
      (BY 2 TO max ch n  -1 DO    print((space, ".")) OD;
        print(new line);
```

```
      TO (ODD max ch n | max ch n -1 | max ch n) DO
           print(space) OD;
      FROM char number(standout) BY -2 TO 3 DO
      print((backspace, ".", backspace, backspace)) OD;
      print(new line);
      BY 2 TO max ch n -1 DO
      print((space, space, backspace, backspace, space, ".")) OD;
      print(new line)
    )
  END;

  print((new line, "A check on lines and pages", newline));
  BEGIN
    PROC print lp = VOID:
      print(("Line number", line number(standout),
            ", page number", page number(standout), ".", new line));
    print lp;
    print(new line);
    print lp;
    print(new page);
    print lp
  END
END
```

- - . - -

```
#smio05#
BEGIN
# 10/08/73, R van Vliet; 30/09/75, revised.
Test the print and putroutines.#

 # Assumes pages more than twice as wide as they are high.#

   INT max ch n = # actual max char, formerly
       max char[standout channel] #
   (FILE f:= standout; INT i;
       on line end(f, (REF FILE f)BOOL: GOTO out);
       DO put(f, space) OD;
   out: i:= char number(f) -1; TO i DO put(f, backspace) OD; i);

  print(("Some tests on PROC(REF FILE)VOID''s", newline));

   ( PROC triangle =(REF FILE f)VOID:
     BEGIN
       FILE rf:=( line number(f) = 1 AND char number(f) = 1 | f
                | FILE ff:=f; on page end(ff,
                           (REF FILE f)BOOL : GOTO out);
                ff);
       PROC nlp =(REF FILE f) VOID:
         new line(f);
```



```
         nlp(rf);
         INT half width = max ch n OVER 2;
         INT i:=1;
         FOR k FROM half width -1 BY -1 TO 0
         DO
           TO k DO space(rf) OD;
           TO i DO put(rf, ".") OD;
           i +:= 2;
           new line(rf)
         OD EXIT
       out: new line(f)
       END;

       print(("First print the full triangle", new page));
       print(triangle);
       print((new line,
         "Now a part of it, to check some administration.", new line,
         "The triangle should be chopped at the end of the page.",
          newline));
       print(triangle);
       print((
          "Now print the triangle as part of a more complicated call."
          new line, triangle,
          "Did it stop at end of page again ?", newline))
   )
END
```

- - . - -

```
#smio06#
BEGIN
# 10/08/73, R van Vliet; 30/09/75, revised.
Test the print and putroutines.#
    INT n dots = 10;

    print(("Print ", whole(n dots, -2), " dots on the next line",
            newline));
    ( PROC spacedot =(REF FILE f) VOID:
      #This procedure is used to print 'n dots' dots in a
      highly recursive call on 'print'.
      First the current position is moved to 'n dots' by printing
      spaces, second the dots are printed from right to left.#
     IF char number(f) < n dots
     THEN space(f); put(f, spacedot)
     ELSE #The spaces are done, now we turn to dotter.#
           put(f, dotter)
     FI,
    PROC dotter =(REF FILE f) VOID:
     IF char number(f) > 1
     THEN put(f, "."); backspace(f);
```

```
      put(f, (backspace, dotter))
    ELSE put (f, ".")
     FI;
     print((spacedot, newline))
  )
END
```

- - . - -

```
#smio07#
BEGIN
# 10/08/73, R van Vliet; 30/09/75, revised.
Test the print and putroutines.#

    print((new line,
      "Print 20 stars and 20 dots alternatingly", new line));
    ( SEMA star allowed =LEVEL 1, dot allowed =LEVEL 0;
      PROC stardot = (REF FILE f) VOID:
      BEGIN
        PROC prstar =(REF FILE f, INT n) VOID:
          IF n NE 1
          THEN prstar(f, n -1); prstar(f, 1)
          ELSE DOWN star allowed; put(f, "*"); UP dot allowed
          FI,
        PROC prdot =(REF FILE f, INT n) VOID:
          IF n NE 1
          THEN prdot(f, n-1); prdot(f, 1)
          ELSE DOWN dot allowed; put(f, "."); UP star allowed
          FI;
        PAR(prstar(f, 20), prdot(f, 20))
          #This parallel call on the recursive procedures 'prstar'
          and 'prdot' should cause the printing of stars and dots,
          looking at the semas before they are actually printed.#
      END;
      print((stardot, newline))
    )
END
```

- - . - -

```
#smio08#
BEGIN
# 10/08/73, R van Vliet; 30/09/75, revised.
Test the print and putroutines.#

    print((newline, "It should print 2:"));
    ( [1:7]PROC (REF FILE) VOID p;
      INT k:=0;
      p[1]:=(REF FILE f) VOID: (k +:=1; SKIP);
```



```
      FOR i FROM 2 TO UPB p DO
        p[i]:=(REF FILE f) VOID: (k +:=1; GOTO l)
      OD;
      print((REF FILE f) VOID:
                FOR i TO UPB p DO p[i](f) OD);
    l:print((k, new line))
    )
END
```

- - . - -

```
#smio09#
( # Parameter of 'print' #

      print((1, 2.0, 3 I 4, "5", "67", TRUE, 16r89, newline));
      print((bytes pack("10"),
            STRUCT(BOOL bo, BITS bi)(TRUE, 4r123123),
            UNION(BOOL, BITS)(4r321321), newline));
      print(([]STRUCT([]REAL rr, INT i)
               (((1.0, 2.0), 3), ((4.0, 5.0), 6)), newline, newline))

(     PROC prent = ([]UNION(INT, REAL, COMPL, BOOL, BITS,
              CHAR, STRING, PROC(REF FILE)VOID) par) VOID:
          FOR i TO UPB par DO print(par[i]) OD;

      prent((1, 2.0, 3 I 4, "5", "67", TRUE, 16r89, newline))
);

( # Parameters of 'printf' #
  printf(($ 3d 1 $, 1, UNION(INT, FORMAT) (2)))
);

( # Badly visible calls of 'print' #
  (print) ((newline, "Parenthesized primary", 1));
  FOR i TO 2
  DO CASE i IN print, write, SKIP ESAC
       ((newline, "Case clause primary", i))
  OD
)

)
```

- - . - -

```
#smio10#
  # Binary transput of structure #
BEGIN

    STRING alphabet = "abcdefghijklmnopqrstuvwxyz";
```

```
    INT size = 1000;
    [ 0 : size] STRUCT(CHAR c, REAL r) str;

    FOR i FROM 0 TO UPB str
    DO str[i]:= (alphabet[ i MOD 26 + 1 ], 1 / (i+1)) OD;

    putbin(standback, str);
    print(("File written", newline));
    reset(standback);
    print(("File rewound",newline));
    getbin(standback, str);
    print(("File read", newline));

    BOOL bad:= FALSE;
    FOR i FROM 0 TO UPB str
    DO bad := bad OR
        IF c OF str[i] /= alphabet[ i MOD 26 + 1 ]
        THEN print(("Char bad in struct ", whole(i,0),
                    ", char is ", c OF str[i],
                    ", char must be ", alphabet[ i MOD 26 + 1 ],
                    newline));
            TRUE
        ELIF r OF str[i] /= 1/(i+1)
        THEN print(("Real bad in struct ", whole(i,0),
                    ", real is ", r OF str[i],
                    ", real must be ", 1/(i+1),
                    newline));
            TRUE
        ELSE FALSE
        FI
    OD;
    IF NOT bad THEN print("Contents of file correct") FI

END
```

- - . - -

```
#smio11#
BEGIN # Simple application of all formats with all allowed modes #

    printf(($ "Integral       " 2(18x 2d) 1$,
        1, LOC INT:= 1));
    printf(($ "Real           " 2(15x 2d.2d) 1$,
        2.0, LOC REAL:= 2.0,
        3, LOC INT:= 3));
    printf(($ "Complex        " 2(7x 2d.2dxix2d.2d) 1$,
        4.0 I 4.0, LOC COMPL:= 4.0 I 4.0,
        5.0, LOC REAL:= 5.0,
        6, LOC INT:= 6));
```



```
    print(newline);
    printf(($ "Boolean        " 2(19x b) 1$,
        TRUE, LOC BOOL:= TRUE));
    printf(($ "String         " 2(16x 4a) 1$,
        "uvwx", LOC STRING:= "uvwx"));
    printf(($ "Character      " 2(19x 1a) 1$,
        STRING("y"), LOC STRING:= "y",
        "z", LOC CHAR:= "z"));
    printf(($ "Bits           " 2(18x 16r 2d) 1$,
        2r10101011, LOC BITS:= 2r10101011));

    print(newline);
    printf(($ "Boolean choice " 2(17x b("cde", "***")) 1$,
        TRUE, LOC BOOL:= TRUE));
    printf(($ "Integral choice" 2(17x c("fgh")) 1$,
        1, LOC INT:= 1));

    print(newline);
    printf(($ "General float  " 2(12x g(8,2,2)) 1$,
        1.0, LOC REAL:= 1.0,
        2, LOC INT:= 2));
    printf(($ "General fixed  " 2(14x g(6,2)) 1$,
        3.0, LOC REAL:= 3.0,
        4, LOC INT:= 4));
    printf(($ "General whole  " 2(18x g(2)) 1$,
        5.0, LOC REAL:= 5.0,
        6, LOC INT:= 6));

    print(newline);
    printf(($ "General        " g 10x, g 1$,
        7, LOC INT:= 7,
        8.0, LOC REAL:= 8.0,
        9.0 I 9.0, LOC COMPL:= 9.0 I 9.0,
        TRUE, LOC BOOL:= TRUE,
        2r1, LOC BITS:= 2r1,
        "w", LOC CHAR:= "w",
        "xyz", LOC STRING:= "xyz"))
END
```

- - . - -

```
#smio12#
BEGIN # Test 'fixed' #

PROC fixed 1 = (REAL v, INT width, after) STRING:
BEGIN

    PROC subfixed =
    #
      This procedure attempts a machine-independent conversion from
```

```
REAL to []CHAR. The conversion will be exact if 'b' is
given a value such that the following conditions hold:

b is integer, x is real
1 < b < maxint OVER 10
if x < 0       then -x > 0
if x < 0       then --x = x
if x >= 1      then x / b * b = x
if 0 < x < 1   then x * b / b = x
if 0 <= x < b  then 0 <= entier(x) < b
if 0 <= x < b  then x - entier(x) < 1
if 0 <= x < b  then x - entier(x) + entier(x) = x
if x > 1       then "ln(x)" * .9 < ln(x) < "ln(x)" * 1.1
               where "ln(x)" is the mathematical log nat of x

Here the arithmetic operators are meant as implemented, the
relational operators are meant in an absolute sense. The text
contains tests for these conditions, where, of course, the
relational operators are implemented by their ALGOL 68 counter-
parts. Discrepancies caused by tests which do not test what they
should test may cause the message "Sloppy arithmetic".

This procedure is not as inefficient as it might have been.
#
    (REAL v, INT after, REF INT point, REF BOOL neg,
     BOOL floating) STRING:
BEGIN
    INT b = 16          # replace by suitable value # ;

# Reports on arithmetic troubles #
    PROC warning =
        (UNION(REAL, INT, CHAR, STRING) l, oper, r, res, ch)
        VOID:
    print((
        "Warning: ", l, oper, r, " is ", res,
        ", should be ", ch, " .", new line
        ));

    PROC sloppy = (REAL x, STRING s, INT lim) VOID:
    print(("Sloppy arithmetic: ", x, " is still ", s,
        " after", lim, " iterations.", newline));

# Determining accuracy #
    # Each decimal operation may cause a loss of at most 1 unit in
      the last decimal. If we now make pessimistic guesses at the
      number of operations, we can calculate the number of extra
      digits needed.
    #

    PRIO LN = 9;
    OP LN = (INT a, REAL b) INT:
```



```
        (b < 1 | 0 | ENTIER(ln(b) / ln(a))) + 1;

    INT max exp  = b LN max_real,
        max mant = b LN (1 / small_real) + 1;

# Floating decimal arithmetic #

    MODE DEC= STRUCT(REF [] INT d, INT p);
    # The value is <d> * 10 ** p, where <d> is d considered as
      a decimal fraction with the point in the (non-existent)
      position 0
    #

    PROC zero = (INT size) DEC:
    BEGIN HEAP [size] INT d;
        FOR i TO UPB d DO d[i]:= 0 OD;
        (d, 0)
    END;

    OP +:= = (REF DEC dc, INT a) VOID:
    BEGIN
        REF [] INT d = d OF dc, REF INT p = p OF dc;
        INT upb = UPB d;
        INT carry:= a, i:= p;

        WHILE carry > 0
        DO
            WHILE i <= 0
            DO
                d[2:]:= d[1: upb - 1]; d[1]:= 0; p +:= 1; i +:= 1
            OD;
            IF i > upb
            THEN carry OVERAB 10
            ELSE
                REF INT di = d[i];
                INT val = di + carry;
                (di:= val MOD 10, carry:= val OVER 10)
            FI;
            i -:= 1
        OD
    END;

    OP *:= = (REF DEC dc, INT a) VOID:
    BEGIN
        REF [] INT d = d OF dc, REF INT p = p OF dc;
        INT upb = UPB d;
        INT carry:= 0, i:= upb;

        WHILE i > 0 OR carry > 0
        DO
            WHILE i <= 0
```

```
        DO
            d[2:]:= d[1: upb - 1]; d[1]:= 0; p +:= 1; i +:= 1
        OD;
        REF INT di = d[i];
        INT val = di * a + carry;
        (di:= val MOD 10, carry:= val OVER 10);
        i -:= 1
    OD
END;

OP /:= = (REF DEC dc, INT a) VOID:
BEGIN
    REF [] INT d = d OF dc, REF INT p = p OF dc;
    INT upb = UPB d;
    INT carry:= 0, i:= 1;

    WHILE (i <= upb | TRUE | carry > 0 AND d[1] = 0)
    DO
        WHILE i > upb
        DO
            d[1: upb -1]:= d[2:]; d[upb]:= 0; p -:= 1; i -:= 1
        OD;
        REF INT di = d[i];
        INT val = di + carry * 10;
        (di:= val OVER a, carry:= val MOD a);
        i +:= 1
    OD
END;

# Actual subfixed #

    neg:= v < 0;

    REAL x:=
        IF neg
        THEN REAL x = -v;
            IF -x /= v
            THEN warning("", "-", x, -x, v)
            FI;
            x
        ELSE v
        FI,
    INT exp:= 0;

    DEC dc:=
        zero( (INT m = 10 LN x + after + 1; m > real_width
              | m | real_width
              )
            + 10 LN REAL(max exp * 1 + max mant * 2) + 1);

    # We keep the following invariant:
```



```
        |v| = (x + |dc|) * b ** exp
    #

    # First we make 'x' zero #
    IF x > 0
    THEN
        TO max exp WHILE x < 1
        DO REAL y = x*b; exp -:=1;
            IF y / b /= x
            THEN warning(y, "/", b, y/b, x)
            FI;
            x:= y
        OD;
        IF x < 1 THEN sloppy(x, "< 1", max exp) FI;
        TO max exp WHILE NOT (x < 1)
        DO REAL y = x/b; exp +:=1;
            IF y * b /= x
            THEN warning(y, "*", b, y*b, x)
            FI;
            x:= y
        OD;
        IF NOT (x < 1) THEN sloppy(x, ">= 1", max exp) FI;

        # Now 1/b <= x < 1 #
        TO max mant WHILE x > 0
        DO
            (x:= x * b, dc *:= b, exp -:= 1);
            INT dig = ENTIER x;
            IF dig < 0
            THEN warning("", "ENTIER", x, dig, ">= 0")
            FI;
            IF dig >= b
            THEN warning("", "ENTIER", x, dig, "< b")
            FI;
            REAL y = x - dig; dc +:= dig;
            IF y >= 1
            THEN warning(x, "-", dig, y, "< 1")
            FI;
            IF y + dig /= x
            THEN warning(y, "+", dig, y+dig, x)
            FI;
            x:= y
        OD;
        IF x > 0 THEN sloppy(x, "> 0", max mant) FI
    FI;
    # Now x = 0, and consequently  |v| = |dc| * b ** exp #

    # Second we make 'exp' 0 #
    WHILE exp > 0
    DO (dc *:= b, exp -:= 1) OD;
    WHILE exp < 0
```

```
    DO (dc /:= b, exp +:=1) OD;

    # Now  |v| = |dc|, i.e. 'v' has been converted to decimal #

    # We shall now fill 's' from 'dc' in the required format #

    OP ELEM = (INT i, REF [] INT d) CHAR:
        "0123456789" [(i < 1| 0 |: i > UPB d | 0 | d[i])+1];

    IF floating
    THEN
        print((newline,"Floating version not implemented",newline));
        SKIP
    ELSE
        REF [] INT d = d OF dc, INT p = p OF dc;
        [UPB d] CHAR s; INT i:= 0;

        FOR k TO p
        DO CHAR ch = k ELEM d;
            IF i = 0 AND ch = "0" THEN SKIP
            ELSE s[i+:=1]:= ch
            FI
        OD;
        point:= i;
        FOR k FROM p + 1 TO p + after + 1
        DO s[i+:=1]:= k ELEM d OD;
        s[1 : point + after + 1]
    FI
END # subfixed # ;

PROC round = (INT k, REF STRING s) BOOL:
IF BOOL carry:= char_dig(s[k+1]) >= 5; s:= s[:k]; carry
THEN
    FOR j FROM k BY -1 TO 1 WHILE carry
    DO INT d = char_dig(s[j]) + 1; carry:= d = 10;
        s[j]:= (carry | "0" | "0123456789"[d+1])
    OD;
    (carry | "1" PLUSTO s); carry
ELSE FALSE
FI;

PROC char_dig = (CHAR c) INT:
(INT i; char_in_string(c, i, "0123456789"); i-1);

# Actual fixed #
IF # no value can be converted legally with these parameters: #
    after < 0
    OR width < 0 AND after > - width - 1
    OR width > 0 AND after >  width - 2
THEN (width = 0 | 1 | ABS width) * error_char
ELIF
```



```
        INT point, BOOL neg;
        STRING s:= subfixed(v, after, point, neg, FALSE);
        STRING sign = (neg | "-" |: width > 0 | "+" | "" );
        width = 0
    THEN
        ( round(UPB s-1, s) | point +:= 1 );
        ( UPB s = 0 | s:= "0"; point:= 1);

        sign + (point = UPB s | s | s[:point] + "." + s[point+1:])
    ELSE
        INT w = ABS width - UPB sign;
        INT tail =
            (INT lim = w - point - 1 + (w=point AND point>0 | 1 | 0);
                (lim < after | lim | after)
            );

        IF tail < 0 THEN ABS width * error_char
        ELSE
            s:= s[ : point + tail + 1];

            ( round(UPB s-1, s) | point +:= 1 );
            ( UPB s = 0 | s:= "0"; point:= 1);

            INT space = w - UPB s - (point = UPB s | 0 | 1);

            IF space < 0 AND tail = 0 THEN ABS width * error_char
            ELSE
                IF space < 0
                THEN s:= s[ : UPB s - 1]
                ELIF space >= 1 AND point = 0
                THEN "0" PLUSTO s; point +:= 1
                FI;
                s:= sign +
                  (point = UPB s | s | s[:point] + "." + s[point+1:])
                (ABS width - UPB s) * " " + s
            FI
        FI
    FI
END # fixed 1 # ;


    # Testing equipment #
    PROC t0 = VOID:
    BEGIN
        FOR v TO UPB vals
        DO REAL value = vals[v];
            t1(value);
            IF value > 0
            THEN t1(DOWN value); t1(UP value)
            FI
        OD;
```

```
    TO 20 DO t4(wild_real) OD;
    t4(max_real)
END;

PROC t1 = (REAL v) VOID:
FOR width FROM -4 TO 9
DO t2(-v, -width); t2(v, -width) OD;

PROC t2 = (REAL v, INT width) VOID:
BEGIN
    FOR after FROM -1 TO 4
    DO t3(v, width, after) OD;
    IF width = 0 THEN t4(v) FI
END;

PROC t3 = (REAL v, INT width, after) VOID:
IF
    STRING s1 = fixed(v, width, after),
           s2 = fixed 1(v, width,after);
    s1 /= s2
THEN
    print((v, whole(width, -4), whole(after, -4), ", is """,
    s1, """, must be """, s2, """""", newline))
FI;

PROC t4 = (REAL v) VOID:
t3(v, 0 ,real_width + 1);

OP DOWN = (REAL x) REAL:
(REAL y := x;
    FOR i WHILE x = y
    DO y:= x * (1 - i * small_real) OD;
    y
);

OP UP = (REAL x) REAL:
(REAL y := x;
    FOR i WHILE x = y
    DO y:= x * (1 + i * small_real) OD;
    y
);

PROC wild_real = REAL:
exp(random * real_width + ln(10));

[] REAL vals =
(
    0.0,
    0.01,
    0.0449,
    0.4449,
```



```
    0.9945,
    9.945,
    99.45,
    100
);

t0
END
```

- - . - -

```
#stan01#
  # Standard operators #
    # Some characters used in Chapter 10 do not exist as such in the
      IFIP Standard Hardware Representation for ALGOL 68 [Boom and
      Hansen]. They are represented here by two crosses followed by
      one Standard Hardware Character. These combinations can be used
      for editing purposes.
      It concerns the following characters:

      or symbol                  ##o
      and symbol                 ##a
      ampersand symbol           ##@
      differs from symbol        ##=
      is at most symbol          ##<
      is at least symbol         ##>
      over symbol                ##:
      window symbol              ##w
      floor symbol               ##f
      ceiling symbol             ##c
      plus i times symbol        ##i
      not symbol                 ##-
      tilde symbol               ##t
      down symbol                ##d
      up symbol                  ##u
      times symbol               ##*

    #

BEGIN # All non-long non-short standard items #

  # 10.2.1. Environment enquiries #

  print(("10.2.1. Environment enquiries", newline));
  print((int lengths, newline));
  print((int shorths, newline));
  print((max int, newline));
  print((real lengths, newline));
  print((real shorths, newline));
  print((max real, newline));
```

```
print((small real, newline));
print((bits lengths, newline));
print((bits shorths, newline));
print((bits width, newline));
print((bytes lengths, newline));
print((bytes shorths, newline));
print((bytes width, newline));
print((ABS "a", newline));
print((REPR 60, newline));
print((max abs char, newline));
print((null character, newline));
print((flip, newline));
print((flop, newline));
print((error char, newline));
print((blank, newline));
print(newline);

# 10.2.2. Standard modes #

(   BOOL b = FALSE;
    INT i = 0;
    REAL x = 0.0;
    CHAR c = "a";
    COMPL z = (0.0, 0.0);
    BITS w = 2r1;
    BYTES v = SKIP;
    STRING s = "";
    SKIP
);

# 10.2.3.1. Rows and associated operations #

[]INT ri = (1);
print(("10.2.3.1. Row and associated operations", newline));
print((1 LWB ri, newline));
print((1 ##f ri, newline));
print((1 UPB ri, newline));
print((1 ##c ri, newline));
print((LWB ri, newline));
print((##f ri, newline));
print((UPB ri, newline));
print((##c ri, newline));
print(newline);

# 10.2.3.2. Operations on boolean operands #

BOOL b = FALSE;
print(("10.2.3.2. Operations on boolean operands", newline));
print((b ##o b, newline));
print((b OR b, newline));
print((b ##a b, newline));
```



```
print((b ##@ b, newline));
print((b AND b, newline));
print((##- b, newline));
print((##t b, newline));
print((NOT b, newline));
print((b = b, newline));
print((b EQ b, newline));
print((b ##= b, newline));
print((b /= b, newline));
print((b NE b, newline));
print((ABS b, newline));
print(newline);

# 10.2.3.3. Operations on integral operands #

INT i = 1;
print(("10.2.3.3. Operations on integral operands", newline));
print((i < i, newline));
print((i LT i, newline));
print((i ##< i, newline));
print((i <= i, newline));
print((i LE i, newline));
print((i = i, newline));
print((i EQ i, newline));
print((i ##= i, newline));
print((i /= i, newline));
print((i NE i, newline));
print((i ##> i, newline));
print((i >= i, newline));
print((i GE i, newline));
print((i > i, newline));
print((i GT i, newline));
print((i - i, newline));
print((- i, newline));
print((i + i, newline));
print((+ i, newline));
print((ABS i, newline));
print((i ##* i, newline));
print((i * i, newline));
print((i ##: i, newline));
print((i % i, newline));
print((i OVER i, newline));
print((i ##:##* i, newline));
print((i ##:* i, newline));
print((i %##* i, newline));
print((i %* i, newline));
print((i MOD i, newline));
print((i / i, newline));
print((i ##u i, newline));
print((i ** i, newline));
print((i UP i, newline));
```

```
print((ODD i, newline));
print((SIGN i, newline));
print((i ##i i, newline));
print((i +##* i, newline));
print((i +* i, newline));
print((i I i, newline));
print(newline);

# 10.2.3.4. Operations on real operands #

REAL x = 1.0;
print(("10.2.3.4. Operations on real operands", newline));
print((x < x, newline));
print((x LT x, newline));
print((x ##< x, newline));
print((x <= x, newline));
print((x LE x, newline));
print((x = x, newline));
print((x EQ x, newline));
print((x ##= x, newline));
print((x /= x, newline));
print((x NE x, newline));
print((x ##> x, newline));
print((x >= x, newline));
print((x GE x, newline));
print((x > x, newline));
print((x GT x, newline));
print((x - x, newline));
print((- x, newline));
print((x + x, newline));
print((+ x, newline));
print((ABS x, newline));
print((x ##* x, newline));
print((x * x, newline));
print((x / x, newline));
print((ROUND x, newline));
print((SIGN x, newline));
print((ENTIER x, newline));
print((##f x, newline));
print((x ##i x, newline));
print((x +##* x, newline));
print((x +* x, newline));
print((x I x, newline));
print(newline);

# 10.2.3.5. Operations on arithmetic operands #

print(("10.2.3.5. Operations on arithmetic operands", newline));
print((x - i, newline));
print((x + i, newline));
print((x ##* i, newline));
```



```
print((x * i, newline));
print((x / i, newline));
print((i - x, newline));
print((i + x, newline));
print((i ##* x, newline));
print((i * x, newline));
print((i / x, newline));
print((x < i, newline));
print((x LT i, newline));
print((x ##< i, newline));
print((x <= i, newline));
print((x LE i, newline));
print((x = i, newline));
print((x EQ i, newline));
print((x ##= i, newline));
print((x /= i, newline));
print((x NE i, newline));
print((x ##> i, newline));
print((x >= i, newline));
print((x GE i, newline));
print((x > i, newline));
print((x GT i, newline));
print((i < x, newline));
print((i LT x, newline));
print((i ##< x, newline));
print((i <= x, newline));
print((i LE x, newline));
print((i = x, newline));
print((i EQ x, newline));
print((i ##= x, newline));
print((i /= x, newline));
print((i NE x, newline));
print((i ##> x, newline));
print((i >= x, newline));
print((i GE x, newline));
print((i > x, newline));
print((i GT x, newline));
print((x ##i i, newline));
print((x +##* i, newline));
print((x +* i, newline));
print((x I i, newline));
print((i ##i x, newline));
print((i +##* x, newline));
print((i +* x, newline));
print((i I x, newline));
print((x ##u i, newline));
print((x ** i, newline));
print((x UP i, newline));
print(newline);

# 10.2.3.6. Operations on character operands #
```

```
CHAR c = "a";
print(("10.2.3.6. Operations on character operands", newline));
print((c < c, newline));
print((c LT c, newline));
print((c ##< c, newline));
print((c <= c, newline));
print((c LE c, newline));
print((c = c, newline));
print((c EQ c, newline));
print((c ##= c, newline));
print((c /= c, newline));
print((c NE c, newline));
print((c ##> c, newline));
print((c >= c, newline));
print((c GE c, newline));
print((c > c, newline));
print((c GT c, newline));
print((c + c, newline));
print(newline);

# 10.2.3.7. Operations on complex operands #

COMPL z = (1.0, 1.0);
print(("10.2.3.7. Operations on complex operands", newline));
print((RE z, newline));
print((IM z, newline));
print((ABS z, newline));
print((ARG z, newline));
print((CONJ z, newline));
print((z = z, newline));
print((z EQ z, newline));
print((z ##= z, newline));
print((z /= z, newline));
print((z NE z, newline));
print((z - z, newline));
print((- z, newline));
print((z + z, newline));
print((+ z, newline));
print((z ##* z, newline));
print((z * z, newline));
print((z / z, newline));
print((z - i, newline));
print((z + i, newline));
print((z ##* i, newline));
print((z * i, newline));
print((z / i, newline));
print((z - x, newline));
print((z + x, newline));
print((z ##* x, newline));
print((z * x, newline));
print((z / x, newline));
print((i - z, newline));
print((i + z, newline));
print((i ##* z, newline));
print((i * z, newline));
print((i / z, newline));
print((x - z, newline));
print((x + z, newline));
print((x ##* z, newline));
print((x * z, newline));
print((x / z, newline));
print((z ##u i, newline));
print((z ** i, newline));
print((z UP i, newline));
print((z = i, newline));
print((z EQ i, newline));
print((z ##= i, newline));
print((z /= i, newline));
print((z NE i, newline));
print((z = x, newline));
print((z EQ x, newline));
print((z ##= x, newline));
print((z /= x, newline));
print((z NE x, newline));
print((i = z, newline));
print((i EQ z, newline));
print((i ##= z, newline));
print((i /= z, newline));
print((i NE z, newline));
print((x = z, newline));
print((x EQ z, newline));
print((x ##= z, newline));
print((x /= z, newline));
print((x NE z, newline));
print(newline);

# 10.2.3.8. Bits and associated operations #

BITS w = 2r1;
print(("10.2.3.8. Bits and associated operations", newline));
print((w = w, newline));
print((w EQ w, newline));
print((w ##= w, newline));
print((w /= w, newline));
print((w NE w, newline));
print((w ##o w, newline));
print((w OR w, newline));
print((w ##a w, newline));
print((w ##@ w, newline));
print((w AND w, newline));
print((w ##< w, newline));
print((w <= w, newline));
```

```
print((w LE w, newline));
print((w ##> w, newline));
print((w >= w, newline));
print((w GE w, newline));
print((w ##u i, newline));
print((w UP i, newline));
print((w SHL i, newline));
print((w ##d i, newline));
print((w DOWN i, newline));
print((w SHR i, newline));
print((ABS w, newline));
print((BIN i, newline));
print((i ELEM w, newline));
print((i ##w w, newline));
print((bits pack((TRUE, FALSE)), newline));
print((##- w, newline));
print((##t w, newline));
print((NOT w, newline));
print(newline);

# 10.2.3.9. Bytes and associated operations #

BYTES v = bytes pack("a");
print(("10.2.3.9. Bytes and associated operations", newline));
print((v < v, newline));
print((v LT v, newline));
print((v ##< v, newline));
print((v <= v, newline));
print((v LE v, newline));
print((v = v, newline));
print((v EQ v, newline));
print((v ##= v, newline));
print((v /= v, newline));
print((v NE v, newline));
print((v ##> v, newline));
print((v >= v, newline));
print((v GE v, newline));
print((v > v, newline));
print((v GT v, newline));
print((i ELEM v, newline));
print((i ##w v, newline));
print((bytes pack("a"), newline));
print(newline);

# 10.2.3.10. Strings and associated operations #

STRING s = "a";
print(("10.2.3.10. Strings and associated operations", newline));
print((s < s, newline));
print((s LT s, newline));
print((s ##< s, newline));
```



```
print((s <= s, newline));
print((s LE s, newline));
print((s = s, newline));
print((s EQ s, newline));
print((s ##= s, newline));
print((s /= s, newline));
print((s NE s, newline));
print((s ##> s, newline));
print((s >= s, newline));
print((s GE s, newline));
print((s > s, newline));
print((s GT s, newline));
print((s < c, newline));
print((s LT c, newline));
print((s ##< c, newline));
print((s <= c, newline));
print((s LE c, newline));
print((s = c, newline));
print((s EQ c, newline));
print((s ##= c, newline));
print((s /= c, newline));
print((s NE c, newline));
print((s ##> c, newline));
print((s >= c, newline));
print((s GE c, newline));
print((s > c, newline));
print((s GT c, newline));
print((c < s, newline));
print((c LT s, newline));
print((c ##< s, newline));
print((c <= s, newline));
print((c LE s, newline));
print((c = s, newline));
print((c EQ s, newline));
print((c ##= s, newline));
print((c /= s, newline));
print((c NE s, newline));
print((c ##> s, newline));
print((c >= s, newline));
print((c GE s, newline));
print((c > s, newline));
print((c GT s, newline));
print((s + s, newline));
print((s + c, newline));
print((c + s, newline));
print((s ##* i, newline));
print((s * i, newline));
print((i ##* s, newline));
print((i * s, newline));
print((c ##* i, newline));
print((c * i, newline));
```

```
print((i ##* c, newline));
print((i * c, newline));
print(newline);

# 10.2.3.11. Operations combined with assignations #

INT ii:= i, REAL xx:= x, COMPL zz:= z, STRING ss:= s;
print(("10.2.3.11. Operations combined with assignations", newline));
print((ii MINUSAB i, newline));
print((ii -:= i, newline));
print((xx MINUSAB x, newline));
print((xx -:= x, newline));
print((zz MINUSAB z, newline));
print((zz -:= z, newline));
print((ii PLUSAB i, newline));
print((ii +:= i, newline));
print((xx PLUSAB x, newline));
print((xx +:= x, newline));
print((zz PLUSAB z, newline));
print((zz +:= z, newline));
print((ii TIMESAB i, newline));
print((ii ##*:= i, newline));
print((ii *:= i, newline));
print((xx TIMESAB x, newline));
print((xx ##*:= x, newline));
print((xx *:= x, newline));
print((zz TIMESAB z, newline));
print((zz ##*:= z, newline));
print((zz *:= z, newline));
print((ii OVERAB i, newline));
print((ii ##::= i, newline));
print((ii %:= i, newline));
print((ii MODAB i, newline));
print((ii ##:##*:= i, newline));
print((ii ##:*:= i, newline));
print((ii %##*:= i, newline));
print((ii %*:= i, newline));
print((xx DIVAB x, newline));
print((xx /:= x, newline));
print((zz DIVAB z, newline));
print((zz /:= z, newline));
print((xx MINUSAB i, newline));
print((xx -:= i, newline));
print((xx PLUSAB i, newline));
print((xx +:= i, newline));
print((xx TIMESAB i, newline));
print((xx ##*:= i, newline));
print((xx *:= i, newline));
print((xx DIVAB i, newline));
print((xx /:= i, newline));
print((zz MINUSAB i, newline));
```



```
print((zz -:= i, newline));
print((zz PLUSAB i, newline));
print((zz +:= i, newline));
print((zz TIMESAB i, newline));
print((zz ##*:= i, newline));
print((zz *:= i, newline));
print((zz DIVAB i, newline));
print((zz /:= i, newline));
print((zz MINUSAB x, newline));
print((zz -:= x, newline));
print((zz PLUSAB x, newline));
print((zz +:= x, newline));
print((zz TIMESAB x, newline));
print((zz ##*:= x, newline));
print((zz *:= x, newline));
print((zz DIVAB x, newline));
print((zz /:= x, newline));
print((ss PLUSAB s, newline));
print((ss +:= s, newline));
print((s PLUSTO ss, newline));
print((s +=: ss, newline));
print((ss PLUSAB c, newline));
print((ss +:= c, newline));
print((c PLUSTO ss, newline));
print((c +=: ss, newline));
print((ss TIMESAB i, newline));
print((ss ##*:= i, newline));
print((ss *:= i, newline));
print(newline);

# 10.2.3.12. Standard mathematical constants and functions #

print(("10.2.3.12. Standard math. constants and functions", newline
print((pi, newline));
print((sqrt(x), newline));
print((exp(x), newline));
print((ln(x), newline));
print((cos(x), newline));
print((arccos(x), newline));
print((sin(x), newline));
print((arcsin(x), newline));
print((tan(x), newline));
print((arctan(x), newline));
print((next random(ii), newline));
print(newline);

# 10.2.4. Synchronization operations #

(print(("10.2.4. Synchronization operations", newline));
    SEMA pv = LEVEL 0;
    PAR((DOWN pv; print(("last", LEVEL pv, newline))),
```

```
            (UP pv; print(("first", LEVEL pv, newline))))
  );

  SKIP
END
```

- - . - -

```
#stan02#
BEGIN # Standard I/O #

  INT i = 1, INT ii := 1, REAL r = 1.0, CHAR c = "a",
  STRING s = "a";

  # 10.3.1.2. Channels #

  CHANNEL ch = stand out channel;
  print(("10.3.1.2. Channels", newline));
  print((estab possible(ch), newline));
  print((estab possible(stand in channel), newline));
  print((estab possible(stand out channel), newline));
  print((estab possible(stand back channel), newline));
  print(newline);

  # 10.3.1.3. Files #

  FILE f := stand out;
  PROC p = (REF FILE f) BOOL : TRUE # event routine #;
  PROC q = (REF FILE f, REF CHAR c) BOOL : TRUE # ch err #;
  print(("10.3.1.3. Files", newline));
  print((get possible(f), newline));
  print((put possible(f), newline));
  print((bin possible(f), newline));
  print((compressible(f), newline));
  print((reset possible(f), newline));
  print((set possible(f), newline));
  print((reidf possible(f), newline));
  print((estab possible(chan(f)), newline));
  print(((make term(f, s); "make term"), newline));
  print(((on logical file end(f, p); "on logical file end"),
                                                    newline));
  print(((on physical file end(f,p); "on physical file end"),
                                                    newline));
  print(((on page end(f, p); "on page end"), newline));
  print(((on line end(f, p); "on line end"), newline));
  print(((on format end(f, p); "on format end"), newline));
  print(((on value error(f, p); "on value error"), newline));
  print(((on char error(f, q); "on char error"), newline));
  IF reidf possible(f)
  THEN print(((reidf(f, s); "reidf"), newline))
```



```
  ELSE print(("no reidf", newline)) FI;
  print(newline);

  # 10.3.1.4. Opening and closing files #

  print(("10.3.1.4. Opening and closing files", newline));
  print((establish(f, "a", ch, 1, 1, 1), newline));
  print((create(f, ch), newline));
  print((open(f, "b", ch), newline));
  print(newline);

  # 10.3.1.5. Position enquiries #

  print(("10.3.1.5. Position enquiries", newline));
  print((char number(f), newline));
  print((line number(f), newline));
  print((page number(f), newline));
  print(newline);

  # 10.3.1.6. Layout routines #

  print(("10.3.1.6. Layout routines", newline));
  print(((space(f); "space"), newline));
  print(((backspace(f); "backspace"), newline));
  print(((newline(f); "newline"), newline));
  print(((newpage(f); "newpage"), newline));
  IF set possible(f)
  THEN print(((set(f, 1, 1, 1); "set"), newline))
  ELSE print(("no set", newline)) FI;
  IF reset possible(f)
  THEN print(((reset(f); "reset"), newline))
  ELSE print(("no reset", newline)) FI;
  print(((set char number(f, 1); "set char number"), newline));
  print(newline);

  # 10.3.2.1. Conversion routines #

  print((whole(r, i), newline));
  print((whole(i, i), newline));
  print((fixed(r, i, i), newline));
  print((fixed(i, i, i), newline));
  print((float(r, i, i, i), newline));
  print((float(i, i, i, i), newline));
  print((char in string(c, ii, s), newline));
  print((int width, newline));
  print((real width, newline));
  print((exp width, newline));
  print(newline);

  # 10.5.1. The particular prelude #
```

```
print(("10.5.1. The particular prelude", newline));
print(((last random:= 1968; random), newline));
print((get possible(stand in), newline));
print((get possible(stand out), newline));
print((get possible(stand back), newline));
write(("write", newline));
print(((read(LOC [1:0] CHAR); "read"), newline));

stop

ND
```

- - . - -

```
stan03#
EGIN # All format items #

    INT i = 2; UNION(INT, REAL) uir = 2;
    print(("10.3.4.1. Literals and insertions", newline));
    printf(($ d 1 $, 1));
    printf(($ # comment # d 1 $, 1));
    printf(($ CO comment CO  d 1 $, 1));
    printf(($ COMMENT comment COMMENT d 1 $, 1));
    printf(($ x 2(d) 1 $, 1));
    printf(($ x k d 1 $, 1));
    printf(($ x x d 1 $, 1));
    printf(($ x y d 1 $, 1));
    printf(($ x p d 1 $, 1));
    printf(($ x q d 1 $, 1));
    printf(($ x "one" d 1 $, 1));
    printf(($ x 2"one" d 1 $, 1));
    printf(($ x "one" 2"two" d 1 $, 1));
    printf(($ x "one""two" d 1 $, 1));
    printf(($ x "one"1"two" 2x "three" d 1 $, 1));
    printf(($ x "one"1"two" 2y "three" d 1 $, 1));
    printf(($ x "aa" n(i)y d 1 $, 1));
    printf(($ x "aa" nBEGIN i ENDy d 1 $, 1));
    printf(($ x "aa" n( TRUE | i )y d 1 $, 1));
    printf(($ x "aa" nIF TRUE THEN i FIy d 1 $, 1));
    printf(($ x "aa" n( i | i, i )y d 1 $, 1));
    printf(($ x "aa" nCASE i IN i, i ESACy d 1 $, 1));
    printf(($ x "aa" n( uir | (INT): i )y d 1 $, 1));
    printf(($ x "aa" nCASE uir IN (INT): i ESACy d 1 $, 1));
    printf(($ x "aa" n(HEAP INT:= i)y d 1 $, 1));
    printf(($ x "Do not show" n(INT:jmp n)y "Do not show" d1 $,1));
mp n:
    printf(($ x "a" d 1 $, 1));
    print(newline);

    print(("10.3.4.2. Integral patterns", newline));
```



```
    printf(($ x d 1 $, 1));
    printf(($ x sd 1 $, 1));
    printf(($ x z 1 $, 1));
    printf(($ x sz 1 $, 1));
    printf(($ x zdzd 1 $, 1));
    printf(($ x z2dzd 1 $, 1));
    printf(($ x z2sdzd 1 $, 1));
    printf(($ x z2sdsz "a"d 1 $, 1));
    printf(($ x z+sdsz "a"d 1 $, 1));
    print(newline);

    print(("10.3.4.3. Real patterns", newline));
    printf(($ x d . d 1 $, 1.0));
    printf(($ x d s. d 1 $, 1.0));
    printf(($ x d "s"s. d 1 $, 1.0));
    printf(($ x d . 1 $, 1.0));

    printf(($ x d . d e + d 1 $, 1.0));
    printf(($ x d . d se + d 1 $, 1.0));
    printf(($ x d . d "a"se + d 1 $, 1.0));
    printf(($ x d . e + d 1 $, 1.0));
    printf(($ x d e + d 1 $, 1.0));
    print(newline);

    print(("10.3.4.4. Boolean patterns", newline));
    printf(($ x b 1 $, TRUE));
    print(newline);

    print(("10.3.4.5. Complex patterns", newline));
    printf(($ x d.d i d.d 1 $, COMPL(1, 1)));
    printf(($ x d.d si d.d 1 $, COMPL(1, 1)));
    printf(($ x d.d "a"si d.d 1 $, COMPL(1, 1)));
    print(newline);

    print(("10.3.4.6. String patterns", newline));
    printf(($ x aa 1 $, "xx"));
    printf(($ x asa 1 $, "xx"));
    printf(($ x 2a 1 $, "xx"));
    printf(($ x a "a"sa 1 $, "xx"));
    print(newline);

    print(("10.3.4.7. Bits patterns", newline));
    printf(($ x 2r d 1 $, 2r1));
    printf(($ x 4r d 1 $, 2r1));
    printf(($ x 8r d 1 $, 2r1));
    printf(($ x 16r d 1 $, 2r1));
    printf(($ x 2r sd 1 $, 2r1));
    printf(($ x "a"2r d 1 $, 2r1));
    print(newline);

    print(("10.3.4.8. Choice patterns", newline));
```

```
    printf(($ x c ( "a", CO c CO 2"a"1"p", "bcd") 1 $, 2));
    printf(($ x "z"c ( "a", CO c CO 2"a"1"p", "bcd") 1 $, 2));

    printf(($ x b ( "a", CO c CO 2"a"1"p") 1 $, FALSE));
    printf(($ x "z"b ( "a", CO c CO 2"a"1"p") 1 $, FALSE));
    print(newline);

    print(("10.3.4.9. Format patterns", newline));
    printf(($ x f ($ d 1 $) $, 1));
    printf(($ x f IF TRUE THEN $ d 1 $ FI $, 1));
    printf(($ x f CASE 1 IN $ d 1 $, SKIP ESAC $, 1));
    printf(($ x f CASE uir IN (INT):$ d 1 $ ESAC $, 1));
    print(newline);

    print(("10.3.4.10. General patterns", newline));
    printf(($ x g 1 $, 1));
    printf(($ x "z"g 1 $, 1));
    printf(($ x g(2) 1 $, 1));
    printf(($ x g(4,1) 1 $, 1));
    printf(($ x g(7,1,2) 1 $, 1));

    printf(($ x g 1 $, 1.0));
    printf(($ x "z"g 1 $, 1.0));
    printf(($ x g(2) 1 $, 1.0));
    printf(($ x g(4,1) 1 $, 1.0));
    printf(($ x g(7,1,2) 1 $, 1.0));

    printf(($ x g(HEAP INT:=7, HEAP INT:=1, HEAP INT:=2)1$,1));
    printf(($ x "Show" g(INT:jmp g) "Do not show" 1 $, 1)); jmp g:
    print(("End of show", newline));

    SKIP
ND
```

- - . - -

```
synt01#
EGIN # Small infringements #
 # All tests are made in separate enclosed clauses to allow
   the parser to recover #

 # No redeclaring of bolds #
 (MODE REAL = INT; SKIP);
 (MODE COMPL = STRUCT(REAL r, phi); SKIP);
 (MODE FILE = INT; SKIP);
 (MODE REF INT = REF INT; SKIP);
 (MODE GOTO = INT; SKIP);
 (MODE GO = INT ; SKIP);
 (MODE IS = INT; SKIP);
 (MODE AT = INT; SKIP);
```



```
  (MODE TRUE = INT; SKIP);
  (MODE EMPTY = INT; SKIP);
  (MODE VOID = INT; SKIP);

  (MODE M = LONG BOOL; SKIP);
  (MODE M = SHORT FILE; SKIP);
  (MODE M = LONG REF INT; SKIP);
  (MODE M = FLEX FLEX [] CHAR; SKIP);

  (MODE LONG BOOL = BITS; SKIP);
  (MODE SHORT FILE = FORMAT; SKIP);
  (MODE LONG REF INT = INT; SKIP);
  (MODE FLEX [] INT = COMPL; SKIP);
  (MODE FLEX M = COMPL; SKIP);

  (PRIO + = 08; SKIP);
  (02r1001, 7r16, 16refg, 17refg);
  (HEAP [@1 : 6]REAL);

  (OP IS = (INT i) BOOL : i = 0; IS 1);
  (OP OF = (INT i) BOOL : i = 0; OF 1);
  (OP AT = (INT i) BOOL : i = 0; AT 1);
  (OP IS = (INT i, j) BOOL : i = j; 1 IS 2);

  # No comments in tags and denotations #
  ( 12  34);   # OK #
  ( 12 # KO # 34);
  ( 12 CO KO CO  34 );
  ( 12 COMMENT KO COMMENT 34 );
  ( algol # KO # 68 );
  (LONG INT i = LENG 1; SKIP);
  (LONG # KO # INT i = LENG 1; SKIP);
  (SHORT # KO # 2r101);

  # GO TO is allowed, but watch the loop-clause #
  (GO TO stop; GO # KO # TO stop; GO
    TO stop);
  (FOR i FROM GO TO stop DO SKIP OD);
  (FROM GO TO stop DO SKIP OD);

  SKIP
END
COMMENT No comment allowed COMMENT
```

- - . - -

```
#synt02#
BEGIN [] REAL x # This is a small program, the rest is missing#
```

- - . - -

```
synt03#
   :

                              - - . - -


synt04#
ND

                              - - . - -


synt05#

                              - - . - -


synt06#
 # Comment without end

                              - - . - -


synt07#
 # Please feel free to shuffle #
one);print((valueOFa,valueOFbs//t,dp*dq*r)END;,m;(k<1;r//:=t;(p
-q)|BEGIN INTr=nOF<0|-i/-j|:j=(RATp,q)<=nOFqCASE FORMu=
eftoperandOFet,v=rightoperandOFet;FORMvalueof=(FORMe)REAL:
ASEeIN(REF CONSTec):valueOFec,(REF VARev):valueOFev,(
EF OP-:==(REF RATp,RATq)REF RAT:p:=p-q,OP*:==(REF RATp
AT CONSTc;valueOFc:=valueOFec-1;c)*udash)ESAC,(REF CALLef
ri,=0|error|INTr=i%j;q)REF RAT:p:=p*q,OP/:==(REF RATp,RAT
REF RAT:p:=p/q;OP=p%nOFq,s,u/v,exp(v*ln(u))ESAC,(REF CALL
):BEGIN REF FUNCTIONf=functionnameOFef;valueOFboundvarOFf=(
Tp,q)BOOL:nINTj)REF INT:iOVERAB BOOL:nOFp*dOFq;(s//t,
*dq*r)END;OP-=(RATp,q)RAT:BEGIN INTnp=nOFp,nq=nOFq%d
p;((nOFp//r)*(nOFq//s),(dOFp//s:=1;valueOFx:=1;f:=a+x/(b+x)
udash=derivativeof(u,x),vdash=derivativeof(v,x);operatorOFetIN
lash+vdash,udash-vdash,u*vdash*dOFp;OP*=(RATp,q)RAT:BEGIN
Tr=nOFp%dOFq,s=nOFq:=dOFq;INTr:=dp%=(RATr)REAL:REAL
OFr)/REAL(dOFr)b:=("b",SKIP),x:=("x",SKIP);valueOFa:=1;
lueOFb|m:=1;1:=k;k:=m);n:m:=kMOD1;(m=0|1|k:=1;1;OP%=(INTi,j)
T:ABS(i=0|j|:j=OFp=nOFqANDdOFp=dOFq;OP/==(RATp,q)
OL:nOFp/=nOFqORdOFp,valueOFx,valueof(derivativeof(g,x))))
D BEGIN MODE RAT=STRUCT(INTn,d);PRIO//=7,p,nq=nOFq;INT
:=dOFp,dq:=dOFq;INTr:=dp%dq;dp;INTdp:=dOFp,dqBEGIN REF
NCTIONf=CALL:=(fdash,g))*derivativeof(g,x)END ESAC;PROC:=
lueof(parameterOFef);valuep//s)*(nOF;OP+=(RATp,q)RAT:
GIN INTnp=nOF 0|i|:i=1ORj=1|1|INTk:=i,1:=j):v*u**(HEAP REF
Ry=boundvarOFf;HEAP FUNCTIONfdash:=(y,derivativeof(bodyOFf,y
;(HEAP:=m;n));OP/=(INTi,j)RAT:(j);OP+:==(REF RATp,RATq
```



```
)REF RAT:p:=p+q,//:==1;OP//=(INTi,j)INT:iOVERj;OP//:==(
REF INTof(bodyOFf)END ESAC;HEAP FORMf,g;HEAP VARa:=("a",
SKIP),%:(q=0|error|:q//r))END/=dOFq;OP<=(RATp,q)BOOL:nOFp
*dOFq<nOFq*dOFp;OP<=+udash*v,(udash-et*vdash)/v,(v|(REF
CONSTec(i//r,j:=(f+one)/(fq<0|-(=dOFp%dOFq;((nOFp//r)*(dOFq//
s),(dOFj;PRIO%=3OFetINu+v,u-v,u*vINTs=np*dq+dp*nq;INTt=s%r;
r//:=t//r));OP VAL(REF CONST):zero,(REF VARev):(ev:=:x|one|
zero),(REF TRIPLEet):dq;dp//:=r;dq//:=r;INTs=np*dq-dp*nq;INTt=s)
*(dOFq//r))END;OP/=(RATp,q)RAT//:=r;dq//:=r;TRIPLEet):
CASE REALu=valueof(leftoperandOFet),v=valueof(rightoperandOFet);
operatorfunctionnameOFef,FORMg=parameterOFef;

//:==(REF INTi,INTj)REF INT:iOVERABj;PRIO%=3;OP%=(INTi,
j)INT:ABS(i=0|j|:j=0|i|:i=1ORj=1|1|INTk:=i,1:=j,m;(k<1|mt,dp*dq
*r)END;OP-=(RATp,q)RAT:BEGIN INTnp=nOFp,nq=nOFq;INTdp
:=dOFp,dq:=dOFq;INTr:=dp%dq;dp//:=r;dq//:=r;INTsRAT:p:=p*q,
OP/:==(REF RATp,RATq)REF RAT:p:=p/q;OP==(RATp,q)BOOL:n
OFp=nOFqANDdOFp=dOFq;OP/==(RATp,q)BOOL:nOFp/=nOFq
ORdOFp/=dOFq;OP<=(RATp,q)BOOL:nOFp*dOFq<nOFq*dOFp;
OP<==(:=1;1:=k;k:=m);n:m:=kMOD1;(m=0|1|k:=1;1:=m;n));OP/=(INTi,
j)RAT:(j<0|-i/-j|:j=0|error|INTr=i%j;(i//r,j//r));OP VAL=(RATr
)REAL:REAL(nOFr)/REAL(dOFr);OP+=(RATp,q)RAT:OFet,v=
rightoperandOFet;FORMudash=derivativeof(u,x),vdash=derivativeof(v,x
);operatorOFetINudash+vdash,udash-vdash,u*vdash+udash*v,(udash-et*
vdash)/v,(v|(REF CONSTec):v*u**(HEAP CONSTc;valueOFc:=valueOF
ec-1;c)*udash)ESAC,(REF CALLef):BEGIN REFnOFp//r)*(nOFq//s)
,(dOFp//s)*(dOFq//r))END;OP/=(RATp,q)RAT:(q=0|error|:q<0|-(
p/-q)|BEGIN INTr=nOFp%nOFq,BEGIN INTnp=nOFp,nq=nOFq;INT
dp:=dOFp,dq:=dOFq;INTr:=dp%dq;dp//:=r;dq//:=r;INTs=np*dq+dp*nq;
INTt=s%r;r//:=t;(s//RATp,q)BOOL:nOFp*dOFq<=nOFq*dOFp;s=d
OFp%dOFq;((nOFp//r)*(dOFq//s),(dOFp//s)*(nOFq//r))END);
OP+:==(REF RATp,RATq)REF RAT:p:=p+q,OP-:==(REF RATp,
RATq)REF RAT:p:=p-q,OP*:==(REF RATp,RATq)REF FUNCTIONf=
functionnameOFef,FORMg=parameterOFef;REF VARy=boundvarOFf;
HEAP FUNCTIONfdash:=(y,derivativeof(bodyOFf,y));(HEAP CALL:=(
fdash,g))*derivativeof(g,x)END ESAC;PROCvalueof=(FORMe)REAL:
CASEeIN(REF CONSTec):valueOFec,(REF VARev):valueOFev,(
REF TRIPLEet):CASE REALu=valueof(leftoperandOFet),v=valueof(
rightoperandOFet);operatorOFetINu+v,u-v,u*v,u/v,exp(v*ln(u))
ESAC,(REF CALLef):BEGIN REF FUNCTIONf=functionnameOFef;value
OFboundvarOFf:=valueof(parameterOFef);valueof(bodyOF(REF
CONST):zero,(REF VARev):(ev:=:x|one|zero),(REF TRIPLEet):CASE
FORMu=leftoperandf)END ESAC;HEAP FORMf,g;HEAP VARa:=("a",
SKIP),b:=("b",SKIP),x:=("x",SKIP);valueOFa:=1;valueOFb:=1;
valueOFx:=1;f:=a+x/(b+x);g:=(f+one)/(f-one);print((valueOFa,value
OFb,valueOFx,valueof(derivativeof(g,x))))END BEGIN MODE RAT=
STRUCT(INTn,d);PRIO//=7,//:==1;OP//=(INTi,j)INT:iOVERj;
OP=np*dq-dp*nq;INTt=s%r;r//:=t;(s//t,dp*dq*r)END;OP*=(RATp,q)
RAT:BEGIN INTr=nOFp%dOFq,s=nOFq%dOFp;((
```

- - . - -

```
#synt08#
  # Straight from the key punch; where does it get stuck? #
BEGIN
FILE program; # contains the program#
establist(program,"program", z type channel,1,10000,80);
FILE result; # will centain the mineed program#
establish(result,"result", z type channel, 1810000,80);
INT line width = 72;
INT c pos:= 0, STRING line;
CHAR quote= """", bold= REPR 39 # apostrophe #,
PROC in item STRING:
(STRING st= in item or comment;
comment(st) | skip comment(st); in item| st);
>PROC comment= (STRING s) BOOL:
s= "#" OR s= bold + "co" OR s= bold + "co" + bold
OR s= bold + "comment" FOR s= bold + "comment" + bold;
PROC skip comment= (STRING s) BOOL:
WHILE in item 2/= s DO OD;
PROC in item2= STRING:
BEGIN more real input; CHAR ch= line[c pos];
    STRUCT(STRING item, INT new pos) res:=
    IF letter(ch)
    THEN INT p= last(letgit);
        (line[c pos: p], p+1)
    ELIF ch= quote
    >THEN INT p= last ((CHAR c) BOOL: cf= quote);
        (line[cpos: p] q quote, p+2)
    ELIF digit(ch)
    THEN INT p= last (digit);
        nline[c pos: p], p + 1)
    >ELIF ch = bold
    THEN INT p= last (letgit);
        (line[c pos: pb q bold,
        p q
        (p = UPB line| 1 |: line[p+1] = bold| 2 | 1))
    ELIF indicant (ch)
    >THEN INT p = last (indicant);
        (line[c pos: p], p + 1)
    ELSE (line[c pos], cpos + 1)
    FI;
    c pos:= new pos OF res; item OF res
END # in item 2 #;
PROC last = (PROC (CHAR) BOOL cond) INT:
(INT p:= cpos;
    FOR d FROM cpos + 1 TO UPB line WHILE cond(line[d])
    DO p=: d >OD ;
    p
);
PROC letter= (CHAR ch) BOOL: "a" <= ch AND ch <= "z";
PROC digit= (CHAR ch) BOOL: "0" <= ch AND ch < = "9";
PROC letgit = (CHAR ch) BOOL: letter (ch) OR digit (ch);
```



```
PROC indicant = (CHAR ch) BOOL:
    char in string (ch, "+-*/=<>:", LOC INT);
>PROC move real input = VOID:
(skip: cpos + := 1;
    IF c pos > UPB line THEN get line; skip FI;
    IF line [cpos]= " " THEN skip FI
);
>PROC get line = VOID:
nget(program, nnewline, line));
    >if UPB line > line width
    THEN line:= line [1: linewidth] FI;
    cpos:= 0
);
>PROC out item= (STRING s) VOID:
(IF char pos (result) + UPB s > line width
    >THEN newline (result) FI;
    put(result, s)
);
# reading the program text #
MODE TEXT = STRUCT (STRING string, REF TEXT next);
>REF TEXT no text = NIL;
REF TEXT first text:= no text, last text:= no text;
on logical file end (program, (REF FILE f) BOOL rum);
#initialize # get(program, line);
DO # until end-of-file # STRING st= in item;
    last text:=
    (last text:=: no text| first text| next >OF last text):=
    HEAP TEXT= (st, no text)
OD;
run:
DO # until input exhausted # INT mean=
    (INT i; read(i); i);
    MODE CHUNK= STRUCT (STRUCT (INT length, REF TEXT text)
        chunk, REF CHUNK next);
    REF CHUNK no chunk = NIL;
    REF CHUNK first chunk = no chun, last chunk = no chunk;
    INT n chunks:= 0; last text:= first text;
    WHILE last text :/=; no text
    DO INF cnt:= 0, REF TEXT p:= last text;
        TO range (2 * mean -1)
        DO (p:/=: no text
            y p;= next >OF p; cnt +:=1)
        >OD # determine chunk #;
        # enter into chunk chain #
        last chunk:=
        (last chunk:=: no chunk
        y first chunk
        | next OF last chunk):=
        HEAP >CHUNK ;= nncnt, last text), NIL);
        n chunk +:= 1; last text:= p
    OD # chunk chain ready #;
```

```
    # tie full-circle #
    next OF last chunk:= first chunk;
    # mix the chunks #
    FOR length FROM n chunks BY -1 TO 1
    >DO TO range (length)
        DO first chunk:= next OF first chunk OD;
        # random chunk found, now write it #
        REF TEXT pd= text OF chunk OF next OF first chunk;
        >TO length OF chunk OF next OF first chunk
        DO out item (string OF p);
            p:= next OF p
        >OD ;
        # remove chunk #
        next FF first chunk:=
            next OF next OF first chunk
        >OD ;
    newline(resultef close (result);
    printf(($"produced" 4 zdx, "chunks of mean length"
        3zd1$, n chunks, mean));
    open (result, "result", z type channel)
 END
END
```

- - . - -

```
#misc01#
BEGIN

  # Comments #
  print("Should print 1:");
  INT i:= 1; #huppeldepupCO i:= 2; CO puppup # print(i) #1# ;
  print(newline);

  #Denotation, test precision#
  IF 3.14159265358979323846264338327 /= pi
  THEN print("3.14159265358979323846264338327 /= pi")
  FI;
  print(newline);

  #Denotations #
  print(
  ( "0 to 3:", 0, 1, 02, 00000000000000000003, newline,
   01.02, .0102, 01.02e0, .0102e0, newline,
   01
   0   .
   2e
   -0    1,
   0102e-04, 01.02e-00000000000000000000000000, .0102e+00, newline,
   "T", "h", "e", " above two lines ", "should be id"
   "entical. ", """did that work?"""))
```



```
END
```

- - . - -

```
#misc02#
( # Format denotation ? #
  FORMAT f =
    (# cp # 1, # count # 0, # bp # 1, # c # () )
    # or something else likely to fool your compiler # ;
  putf(stand out, f)
)
```

- - . - -

```
#misc03#
(   # ALGOL 68 program
      (test on readability of error messages)
      10-7-'73, J. Admiraal #

    PROC inprod = ( [] REAL a, b) REAL:
    (   REAL s:= 0;
        FOR k FROM LWB a TO UPB a DO
        s +:= a[k] * b[k] OD; s ) # inprod # ;

    PROC vecvec = (INT low, up, shift, []REAL a, b) REAL:
    inprod(a[low:up], b[low+shift : up+shift])  # vecvec #;

    PROC matvec = (INT low, up, i, [,] REAL a,
                                        [] REAL b) REAL:
    inprod( a[i,low:up], b[low:up]) # matvec #;

    PROC tamvec = (INT low, up, i, [,] REAL a, [] REALb) REAL:
    inprod( a[low:up, i], b[low:up]) # tamvec #;

    PROC matmat = (INT low, up, i, j, [,] REAL a, b) REAL:
    inprod( a[i, low:up], b[low:up, j]) # matmat #;

    PROC tammat = (INT low, up, i, j, [,] REAL a, b) REAL:
    inprod( a[low:up, i], b[low:up, j]) # tammat #;

    PROC mattam = (INT low, up, i, j, [,] REAL a, b) REAL:
    inprod( a[i, low:up], b[j, low:up]) # mattam #;

# #

    [1 : 10, 1 : 10] REAL ca, [1:10] REAL aa, BOOL bool:= TRUE;

# Twelve bad calls #
    vecvec (1, 10, bool, aa, aa);
```

```
    vecvec (1, 10, ca, aa, aa, 1.0);
    matvec (1, 10, bool, ca, aa);
    tamvec (1, 10, aa, ca, aa);
    matmat (1, 10, 5, 5, aa, bool);
    tammat (1, 10, 5, 5, bool, ca);
    mattam (1, 10, 5, 5, ca, ca, 1.0);
    matmat (bool, 10, 5, 5, ca, ca);
    mattam (aa, 10, 5, 5, ca, ca);
    vecvec (1, 10, 0, aa);
    matvec (bool);
    matmat (1, 10, 5, 5, ca, ca, 1.0)
)
```

- - . - -

```
#misc04#
BEGIN # A primary #

  MODE # M = PROC(INT)[]PROC P, #
         P = REF []PROC(CHAR)N,
         N = [1:0]UNION(INT, CHAR);

  INT ii # partial parameter #;

  FOR k TO 5 DO
  FOR l TO 5 DO
  FOR m TO 2 DO
  print(
    CASE
      BEGIN(INTi)[]PROC P:
        (ii:= i; P: HEAP [1:2] PROC(CHAR)N:=
          (SKIP, (CHAR c) N:(ii,c)))
      END (k)[l][2]("abcde"[l])[m]
    IN (CHAR c): c, (INT i): "12345"[i]
    ESAC
  )OD OD OD
  # hit 'm hard, the output should be :
    1a1b1c1d1e2a2b ...   5c5d5e #
END
```

- - . - -

```
#misc05#
# Runs and yields 1.0, 1.0, 1.0, -2.0 #
BEGIN MODE FORM=UNION(REF CONST,REF VAR,REF TRIPLE,REF
CALL),CONST=STRUCT(REALvalue),VAR=STRUCT(STRINGname,
REALvalue),TRIPLE=STRUCT(FORMleftoperand,INToperator,FORM
rightoperand),FUNCTION=STRUCT(REF VARboundvar,FORMbody),CALL
=STRUCT(REF FUNCTIONfunctionname,FORMparameter);INTplus=1,
```



```
minus=2,times=3,by=4,to=5;HEAP CONSTzero,one;valueOFzero:=0;value
OFone:=1;OP==(FORMa,REF CONSTb)BOOL:CASEaIN(REF CONST
ec):ec:=:bOUT FALSE ESAC;OP+=(FORMa,b)FORM:(a=zero|b|:b=zero|
a|HEAP TRIPLE:=(a,plus,b));OP-=(FORMa,b)FORM:(b=zero|a|HEAP
TRIPLE:=(a,minus,b));OP*=(FORMa,b)FORM:(a=zeroORb=zero|zero|:
a=one|b|:b=one|a|HEAP TRIPLE:=(a,times,b));OP/=(FORMa,b)FORM:(
a=zeroAND NOT(b=zero)|zero|:b=one|a|HEAP TRIPLE:=(a,by,b));OP**
=(FORMa,REF CONSTb)FORM:(a=oneOR(b:=:zero)|one|:b:=:one|a|
HEAP TRIPLE:=(a,to,b));PROCderivativeof=(FORMe,REF VARx)
FORM:CASEeIN(REF CONST):zero,(REF VARev):(ev:=:x|one|zero),
(REF TRIPLEet):CASE FORMu=leftoperandOFet,v=rightoperandOFet;
FORMudash=derivativeof(u,x),vdash=derivativeof(v,x);operatorOFet
INudash+vdash,udash-vdash,u*vdash+udash*v,(udash-et*vdash)/v,(v|(
REF CONSTec):v*u**(HEAP CONSTc;valueOFc:=valueOFec-1;c)*udash
)ESAC,(REF CALLef):BEGIN REF FUNCTIONf=functionnameOFef,
FORMg=parameterOFef;REF VARy=boundvarOFf;HEAP FUNCTIONfdash
:=(y,derivativeof(bodyOFf,y));(HEAP CALL:=(fdash,g))*derivativeof(
g,x)END ESAC;PROCvalueof=(FORMe)REAL:CASEeIN(REF CONST
ec):valueOFec,(REF VARev):valueOFev,(REF TRIPLEet):CASE
REALu=valueof(leftoperandOFet),v=valueof(rightoperandOFet);
operatorOFetINu+v,u-v,u*v,u/v,exp(v*ln(u))ESAC,(REF CALLef):
BEGIN REF FUNCTIONf=functionnameOFef;valueOFboundvarOFf:=
valueof(parameterOFef);valueof(bodyOFf)END ESAC;HEAP FORMf,g;
HEAP VARa:=("a",SKIP),b:=("b",SKIP),x:=("x",SKIP);valueOFa:=
1;valueOFb:=1;valueOFx:=1;f:=a+x/(b+x);g:=(f+one)/(f-one);print((
valueOFa,valueOFb,valueOFx,valueof(derivativeof(g,x))))END
```

- - . - -

```
#misc06#
BEGIN # Test recursion by Ackermann function.
    This program should be run with successive inputs:
    1, 2, 3, 4 and 5,  and will crash at a certain depth.
    See: Y. Sundblad, A Study of the Highly Recursive Ackermann Funct
    as a Test of Recursive Procedures, NA 18, Royal Institute of
    Technology, Stockholm #

    PROC ack = (INT m, n) INT:
    IF m = 0 THEN n + 1
    ELIF n = 0 THEN ack(m-1, 1)
    ELSE ack(m-1, ack(m, n-1))
    FI # ack #;

    INT m;
  #  read(m);# m:= 1;
  #  read(m);# m:= 2;
  #  read(m);# m:= 3;

    FOR n FROM 0 DO
    print( (newline, m, n, ack(m,n), newline ))OD
```

```
ID
```

- - . - -

```
ıisc07#
 # 100 nested proc declarations; prints 100 100 #
:NT i := 0; print(((INT a =
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
'ROC a = INT:
```



```
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
(PROC a = INT:
```

```
PROC a = INT:
PROC a = INT:
PROC a = INT:
PROC a = INT:
PROC a = INT:
PROC a = INT:
+:=1; a);
+:=1; a);
+:=1; a);
+:=1; a);
+:=1; a);
+:=1; a);
+:=1; a);
+:=1; a);
+:=1; a);
+:=1; a);
+:=1; a);
+:=1; a);
+:=1; a);
+:=1; a);
+:=1; a);
-:=1; a);
-:=1; a);
-:=1; a);
-:=1; a);
-:=1; a);
-:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
:=1; a);
```



```
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
i+:=1; a);
```

```
+:=1; a);
+:=1; a);
+:=1; a);
+:=1; a);
), newline));
rint((i, newline)))
```

- - . - -

```
numr01#
EGIN CO This program calculates (x+(1-x)) ** n (which is 1, more or
   less) for various values of x and n, through binomial expansion.
   An attempt has been made to localize the precision of the floating
   point calculations.
   CO

 FOR n TO 100
 DO CO Begin of length package CO
   MODE REEL = # LONG # REAL;

   OP * = (REEL x, INT i) REEL: x * LENG i;
   OP / = (REEL x, INT i) REEL: x / LENG i;

   REEL one = # LONG # 1.0;

   REEL val1 = # LONG # 0.99,
        val2 = # LONG # 0.9,
        val3 = # LONG # 0.5;

   CO end of length package CO

   PROC poly = (REEL x, INT n) REEL :
     (REEL s:= one, a:= one;
       FOR i TO n
       DO a:= a * (one - x) * (n - i + 1) / i;
           s:= s * x + a
       OD;
       s
     ) CO poly CO;

   print((n, poly(val1, n), poly(val2, n), poly(val3, n), newline))
 OD

ND
```

- - . - -


```
#numr02#
BEGIN  # Brinkmanship #

   # What minreal is the smallest positive real on your installation ? #

   PROC test = (STRING titel, REAL min real) VOID:
       (print((newline,newline,titel,newline));
        print(("minreal =           ", minreal, newline));
        print(("2*minreal =         ", 2*minreal, newline));
        print(("minreal*maxreal =   ", minreal*maxreal, newline));
        print(("sqrt(minreal) =     ", sqrt(minreal), newline));
        print(("sqrt(minreal)**2 =  ", sqrt(minreal)*sqrt(minreal),
                                                          newline));
        IF minreal>0
        THEN print(("ln(minreal) =       ", ln(minreal), newline));
             print(("exp(ln(minreal) =   ", exp(ln(minreal)),newline))
        FI);

   test("minreal = 5.0/(smallreal**2 * maxreal)",
        5.0/(smallreal*maxreal*smallreal));
   test("minreal = 2.0/(smallreal**2 * maxreal)",
        2.0/(smallreal*maxreal*smallreal));
   test("minreal = 1.5/(smallreal**2 * maxreal)",
        1.5/(smallreal*maxreal*smallreal));
   test("minreal = 1.1/(smallreal**2 * maxreal)",
        1.1/(smallreal*maxreal*smallreal));
   test("minreal = 1.0/(smallreal**2 * maxreal)",
        1.0/(smallreal*maxreal*smallreal));

   test("minreal = (x; x>0, x/10 = 0)",
        (REAL x:=1; WHILE x/10 > 0 DO x/:=10 OD; x));
   test("minreal = (x; x>0, x/2 = 0)",
        (REAL x:=1; WHILE x/2 > 0 DO x/:=2 OD; x));

   test("minreal = 1/maxreal", 1/maxreal);

   # to compare #

   print((newline,newline,"to compare",newline));
   print(("maxreal =           ", maxreal,newline));
   print(("smallreal =         ", smallreal, newline));
   print(("1/smallreal**2 =    ", 1/(smallreal*smallreal),newline));

   # What would you like this one to do? #

   print((newline,newline,"e**-maxreal is positive",newline));
   print(("exp(-maxreal) = ",exp(-maxreal),newline))

END
```

- - . - -

```
#numr03#
BEGIN # Test ALGOL 68 version of 'zeroin' (MCA 2310 in 'ALGOL 60
    Procedures in Numerical Algebra' by Th.J. Dekker) #

    PROC zero in = (REF REAL x, y, PROC(REAL)REAL f, tol)
    BOOL:
    BEGIN REAL a:= x, b:= y; REAL fa:= f(a), fb:= f(b);
        REAL c:= a, fc:= fa;
        WHILE
            ( ABS fc < ABS fb | # interchange: #
                (a:=b, fa:= fb); (b:=c, fb:= fc); (c:=a, fc:= fa));
            REAL tolb:= tol(b), m:= (c + b) * .5;
            ABS (m - b) > tolb
        DO
         REAL p:= (b - a) * fb, q:= fa - fb;
            (p < 0 | (p:= -p, q:= -q));
            (a:= b, fa:= fb);
            fb:= f(b:=
                IF p <= ABS q * tolb
                THEN (c > b | b + tolb | b - tolb)
                ELIF p < (m - b) * q
                THEN p / q + b
                ELSE m
                FI);
            IF ABS(SIGN fb + SIGN fc) = 2
            THEN (c:= a, fc:= fa) FI
        OD  # while, do # ;
        (x:= b, y:= c); ABS(SIGN fb + SIGN fc) < 2
    END # zero in # ;

##

    REAL eps = 3 * small real;

    PROC test = (REAL x0, y0, PROC(REAL)REAL f, STRING s,
        UNION (STRING, REAL) sol)
    VOID:
    print((newline, "Expression:  ", s, newline,
        "Zero to be found between ", x0, "    and ", y0, newline,
        IF REAL x, y;
            zero in(x:= x0, y:= y0, f,
                (REAL p) REAL : eps + eps * ABS p)
        THEN STRUCT(STRING s1, REAL r1, STRING s2, REAL r2)
             ("Value found at   ", x, "    is ", f(x))
        ELSE "    no solution found"
        FI, newline,
        "Result on EL-X8: ", sol, newline)) #test #;

    test(-1, 0, (REAL x) REAL : exp(x) - x * x, "exp(x) - x * x",
        -0.7034674224979);
    test(1, 10, (REAL x) REAL : ln(x) - x + 2, "ln(x) - x + 2",
```



```
        3.146193220622);
    test(0,  5, (REAL x) REAL : x * x - 4, "x * x - 4", 2.0);
    test(1,1.5, (REAL x) REAL : sin(3 * x), "sin(3 * x)",
        1.047197551197);
    test(-1, 1, (REAL x) REAL : x * x + 1, "x * x + 1",
        "no solution found")
END
```

- - . - -

```
#numr04#
BEGIN # Two versions of the integration procedure 'qad',
    one fully recursive (and understandable), one half-recursive,
    a result of an optimization attempt on the ALGOL60 version #

  REAL eps = 10000 * small real;

  PROC qad fr = (REAL a, b, PROC (REAL) REAL fx,
                 STRUCT (REAL re, ae, REF INT skip) e) REAL:
  BEGIN REAL sum:= 0;
    REAL re = re OF e, ae = ae OF e * 180 / ABS(b - a),
    REF INT skip = skip OF e:= 0;
    REAL h min = ABS(b - a) * re;

    PROC int = (REAL x0, f0, x2, f2, x4, f4) VOID:
      IF REAL x1 = (x0 + x2) / 2, x3 = (x2 + x4) /2;
        REAL  f1 = fx(x1), f3 = fx(x3);
        REAL h = x4 - x0,
          aid1 = 4 * (f1 + f3), aid2 = f0 + f4,
          v = (aid1 + 2 * f2 + aid2) * 15,
          t = 6 * f2 - aid1 + aid2;
          ABS t < ABS v * re + ae
      THEN sum +:= h * (v - t)
      ELIF ABS h < h min THEN skip +:= 1
      ELSE int(x0, f0, x1, f1, x2, f2);
           int(x2, f2, x3, f3, x4, f4)
      FI # of int #;

    int(a, fx(a), (a + b) / 2, fx((a + b) / 2), b, fx(b));
    sum / 180
  END #of qad fr#;

  PROC qad hr = (REAL a, b, PROC (REAL) REAL fx,
                 STRUCT (REAL re, ae, REF INT skip) e) REAL:
  BEGIN REAL x0:= a, f0:= fx(a), x2:= b, f2:= fx(b);
    REAL x1:= (x0 + x2) / 2; REAL f1:= fx(x1);
    REAL sum:= 0;

    REAL re = re OF e, ae = ae OF e * 180 / ABS(b - a),
    REF INT skip = skip OF e:= 0;
```

```
  REAL h min = ABS(b - a) * re;

  PROC int = VOID:
    BEGIN REAL x4 = x2, f4 = f2;
      x2:= x1; f2:= f1;
    anew:
      IF x1:= (x0 + x2) / 2; f1:= fx(x1);
        REAL  x3= (x2 + x4) / 2; REAL f3:= fx(x3);
        REAL h = x4 - x0,
          aid1 = 4 * (f1 + f3), aid2 = f0 + f4;
        REAL v = (aid1 + 2 * f2 + aid2) * 15,
          t = 6 * f2 - aid1 + aid2;
          ABS t < ABS v * re + ae
      THEN sum +:= h * (v - t)
      ELIF ABS h < h min THEN skip +:= 1
      ELSE int; x2:= x3; f2:= f3; GOTO anew FI;
    x0:= x4; f0:= f4
  END #of int#;

  int; sum / 180
END #of qad hr#;

PROC test qad =
  (STRING type,
    PROC (REAL, REAL, PROC (REAL) REAL,
            STRUCT (REAL re, ae, REF INT skip))REAL qad
  ) VOID:
BEGIN INT real size = real width + exp width + 6;
  print((
    newline, "Results for ", type, ":", newline, " ",
    "exponent", (real size - 8) * " ",
    "integral", (real size - 8) * " ",
    "error", (real size - 5) * " ",
    "skip points   time",
    newline));

  PROC exp test = (REAL power, answer) VOID:
  BEGIN
    INT skip, eval:= 0;
    REAL time:= clock;
    REAL result =
      qad(0, 1, (REAL x)REAL: (eval +:= 1;
                  (x <= 0 | 0 | exp(ln(x) * power))),
          (eps, eps, skip)
        );
    time:= clock - time;
    print(( power, ",", result, ",", result-answer, ",",
            whole(skip, -6), ",", whole(eval, -6), ",",
            time, newline))
  END # test exp #;
```



```
    FOR k FROM 4 TO 10 DO exp test(k, 1 / (k+1)) OD;
    FOR k FROM 2 TO 7 DO exp test(1 / k, k / (k+1)) OD
  END # test qad #;

  test qad("fully recursive version", qad fr);
  test qad("half-recursive version", qad hr)

END
```

- - . - -

```
#numr05#
BEGIN # JKok, 730620, test least squares procedures,
    740919, tested on Control Data A68 Compiler, results OK #

  MODE TOLS = STRUCT (REAL prec, max),

  OP * = ([]REAL a, b) REAL :
    (REAL s:= 0; FOR i TO UPB a DO s +:= a[i] * b[i]OD; s),

  OP * = (REAL a, []REAL b) []REAL :
    ([1 : UPB b]REAL c;
      FOR i TO UPB b DO c[i]:= a * b[i]OD; c
    ),

  OP +:= = (REF[]REAL a, []REAL b) REF[]REAL :
    (FOR i TO UPB a DO a[i] +:= b[i] OD; a);

  PROC lsqdec = (REF[,]REAL a, REF TOLS aux,
     REF[]REAL aid, REF[]INT ci) INT :
  IF INT n = 1 UPB a, m = 2 UPB a;
     UPB aid /= m OR UPB ci /= m THEN - 1
  ELSE INT r:= 0, minmn:= (m < n | m | n), pk:= 1,
      REAL w, eps, sigma:= 0, aidk, beta, [1 : m]REAL sum;

      FOR k TO m
      DO IF (w:= sum[k]:= a[ ,k] * a[ ,k]) > sigma
         THEN sigma:= w; pk:= k FI
      OD;

      w:= max OF aux:= sqrt(sigma); eps:= (prec OF aux) * w;
      FOR k TO minmn WHILE w > eps
      DO REAL akk= a[k,pk]; r:= k; ci[k]:= pk;
         IF pk /= k
         THEN []REAL h= a[ ,k]; a[ ,k]:= a[ ,pk];
            a[ ,pk]:= h; sum[pk]:= sum[k]
         FI;
         aidk:= aid[k]:= (akk < 0 | w | - w); a[k,k]:= akk - aidk;
         beta:= - 1 / (sigma - akk * aidk); pk:= k; sigma:= 0;
        FOR j FROM k + 1 TO m
```

```
      DO a[k : ,j] +:= beta * (a[k: ,k] * a[k: ,j]) * a[k: ,k];
         IF (w:= sum[j] -:= a[k,j] ** 2) > sigma
         THEN pk:= j; sigma:= w FI
      OD;
      w:= sqrt(sigma)
    OD;
    r
FI # end of householder triangularization # ,

PROC lsqsol = ([,]REAL a, []REAL aid, [] INT ci, []REAL b)
      []REAL:
BEGIN INT n = 1 UPB a, m = 2 UPB a, INT cik;
  [1:n]REAL bb:= b;

  IF m <= n
  THEN FOR k TO m DO bb[k: ] +:=
     a[k: ,k] * bb[k: ] / (aid[k] * a[k,k]) * a[k: ,k] OD;
     FOR k FROM m BY - 1 TO 1 DO bb[k] :=
        (bb[k] - a[k,k+1: ] * bb[k+1:m]) / aid[k] OD;
     FOR k FROM m - 1 BY - 1 TO 1
     DO IF cik:= ci[k]; cik /= k
        THEN REAL w= bb[k]; bb[k]:= bb[cik]; bb[cik]:= w FI
     OD
  FI;
  bb
END # of computation of least squares solution #;

FOR n FROM 4 TO 6 DO FOR m TO n
DO [1:n,1:m]REAL a, [1:n]REAL b, [1:m]REAL aid,
  [1:m] INT piv, TOLS aux;

  FOR i TO n DO FOR j TO m DO a[i,j]:= i**(j-1)OD OD;
  FOR i TO n DO b[i]:= i**(n-1)OD; prec OF aux:= 1e-10;
  print(newline); print("n ="); print(n);
  print(newline); print("m =");
  print(m); print(new line);
  IF lsqdec(a, aux, aid, piv) < m THEN
      print(" rank < number of columns")
  ELSE [1 : n]REAL sol:= lsqsol(a, aid, piv, b);
      print(" solution :"); FOR k TO m DO
      print(sol[k]) OD;
      print(newline);
      print(" residue : ");
      print(sol[m+1 : ]*sol[m+1 : ]);
      print(newline); print(newline)
  FI

  # Output approximately:
sol: 25.0  res: 2390.0
sol:  -27.0  20.8  res:  226.8
sol:  10.5  -16.7  7.5  res:  1.8
```



```
sol:  0.0  0.0  0.0  1.0  res:  0.0
sol:  195.8  res:  271290.8
sol:  -250.6  148.8  res:  49876.4
sol:  158.4  -201.77  58.43  res:  2081.83
sol:  -43.2  81.43  -49.57  12.0  res:  8.23
sol:  0.0  0.0  0.0  0.0  1.0  res:  0.0
sol:  2033.5  res:  46529717.5
sol:  -2860.0  1398.14  res:  12320657.14
sol:  2250.0  -2434.36  547.5  res:  1129757.14
sol:  -1040.0  1704.25  -823.3  130.56  res:  25257.14
sol:  220.0  -465.75  344.17  -114.44  17.50  res:  57.14
sol:  0.0  0.0  0.0  0.0  0.0  1.0  res:  0.0    #
  OD OD
END
```

- - . - -

```
#numr06#
BEGIN # Vector calculus #

  MODE VEC = [1:1] REAL;

  OP * = (REAL a, VEC b) VEC:
     ([1:UPB b] REAL c;
      FOR i TO UPB b DO c[i]:= a * b[i]OD; c),

  OP * = (VEC a, b) REAL:
     (REAL s:= 0; FOR i TO UPB a DO s+:=a[i]*b[i]OD; s),

  OP + = (VEC a, b) VEC:
     ([1:UPB b] REAL c;
        FOR i TO UPB a DO c[i]:= a[i] + b[i]OD ; c
     );

  PROC dec = (REF [,] REAL a, REF [] INT p) VOID:
  BEGIN INT n = UPB p; INT pk, REAL max, s, [1:n] REAL v;
    FOR i TO n DO v[i]:= 1/sqrt(a[i,] * a[i,])OD  ;
    FOR k TO n
    DO max:= 0; pk:= k;
      FOR i FROM k TO n
      DO a[i,k] -:= a[i, :k-1] * a[ :k-1,k];
        s:= ABS a[i,k] * v[i];
        IF s > max THEN pk:= i; max:= s FI
      OD;
      p[k]:= pk; IF pk /= k
      THEN [] REAL h = a[pk, ]; a[pk, ]:= a[k, ]; a[k, ]:= h;
         v[pk]:= v[k]
      FI;
      FOR i FROM k + 1 TO n
      DO a[k,i] -:= a[k, :k-1] * a[ :k-1,i] OD;
```

```
      a[k,k+1: # this row may be empty #]:= (1 / a[k,k]) * a[k,k+1: ]
    OD
  END # end decomposition of 'a' # ,

  PROC sol = ([,] REAL a, [] INT p, REF [] REAL b) VOID:
  BEGIN INT n = UPB p;
    FOR k TO n
    DO INT pk = p[k], REAL r = b[k];
      b[k]:= (b[pk] - a[k, :k-1] * b[ :k-1]) / a[k,k];
      IF pk /= k THEN b[pk]:= r FI
    OD;
    FOR k FROM n BY - 1 TO 1
    DO b[k] -:= a[k,k+1: ] * b[k+1: ] OD
  END # end of back substitution of solution into 'b' #;

  FOR n TO 8
  DO [1:n, 1:n] REAL a, aa, [1:n] REAL b, [1:n] INT piv;
    print(newline); print(" n ="); print(n); print(newline);
    FOR i TO n DO FOR j TO n
    DO a[i,j]:= aa[i,j]:= 1 / (i + j - 1) OD OD; # Hilbert-matrix #
    FOR i TO n DO b[i]:= 2 / 2 ** i OD;
    dec(a, piv); sol(a, piv, b);
    FOR i TO n DO print(aa[i, ]*b);
                  print(newline);
                  print(2/2**i);
          # these two should approximately be the same #
                  print((newline,newline))
               OD
  OD
END
```

- - . - -

```
#numr07#
BEGIN #JKok, 730612, test Choleski decomposition#

    OP * = ([]REAL a, b) REAL :
    (REAL s:= 0; FOR i TO UPB a DO s +:= a[i] * b[i] OD;s);

    PROC decsym= (REF [,] REAL a, REF [] INT p, REAL aux)
         INT :
    IF INT n = 1 UPB a;
       2 UPB a /= n OR UPB p /= n THEN 0
    ELSE REAL max:= 0, epsnorm, ukk, uki, aii, INT pk:= 1, r:= 0;

        PROC ichvec= (REF [] REAL a, b) VOID :
            IF INT n= UPB a; n > 0 THEN
                [] REAL h= a; a:= b; b:= h
            FI # interchange two vectors#;
```



```
        FOR k TO n
        DO IF a[k,k] > max THEN max:= a[k,k]; pk:= k FI OD;
        epsnorm:= aux * max;
        FOR k TO n WHILE max > epsnorm
        DO INT k1 = k + 1;
            p[k]:= pk; r:= k;
            IF pk /= k
            THEN ichvec(a[ :k-1,k], a[ :k-1,pk]);
                ichvec(a[k,k1:pk - 1], a[k1:pk - 1,pk]);
                ichvec(a[k,pk + 1: ], a[pk,pk + 1: ]);
                a[pk,pk]:= a[k,k]
            FI;
            ukk:= a[k,k]:= sqrt(max); max:= 0; pk:= k1;
            FOR i FROM k1 TO n
            DO uki:= a[k,i]:= (a[k,i] - a[ :k-1,k]*a[ :k-1,i]) / ukk;
                aii:= a[i,i] -:= uki * uki;
                IF aii > max THEN max:= aii; pk:= i FI
            OD
        OD;
        r
    FI # Choleski decomposition with diagonal pivoting#,

    PROC solsym= ([,] REAL a,[] INT p,REF [] REAL b) VOID:
    IF INT n = 1 UPB a;
       2 UPBa = n AND UPB p = n AND UPB b = n
    THEN INT pk, REAL r;
       FOR k TO n
       DO r:= b[k]; pk:= p[k];
          b[k]:= (b[pk] - a[ :k - 1,k] * b[ :k - 1]) / a[k,k];
          IF pk /= k THEN b[pk]:= r FI
       OD;
       FOR k FROM n BY - 1 TO 1
       DO b[k]:= (b[k] - a[k,k+1: ] * b[k+1: ]) / a[k,k] OD;
       FOR k FROM n BY - 1 TO 1
       DO IF pk:= p[k]; pk /= k
           THEN r:= b[k]; b[k]:= b[pk]; b[pk]:= r FI
       OD
    FI # solution of Choleski decomposed system #;

    print(("Value,  expected,  difference",newline,newline));

    FOR n TO 8
    DO [1:n, 1:n] REAL a, aa, [1:n]REAL b, c, [1:n]INT piv;
        FOR i TO n DO FOR j TO n
        DO a[i,j]:= aa[i,j]:= 1 / (2 * n + 1 - i - j) OD OD;
        FOR i TO n DO b[i]:= 2 ** (n - i) OD;
         IF decsym (a, piv, 1e-13) = n
         THEN solsym(a, piv, b);
            FOR i TO n
            DO
               print((aa[i,] * b, REAL(2 ** (n - i)),
```

```
                    aa[i,] * b - 2 ** (n - i), newline))
          OD
        ELSE print("Coefficients matrix is not positive definite")
        FI;
        print(newline)
     OD
    #Output approximately: 1
                           2  1
                           4  2  1
                           . . .
                           128   64   . . .   1 #
END
```

- - . - -

```
#numr08#
BEGIN#Test sqrt#

   PROC warn = (STRING s)VOID:
        BEGIN
           print((newline,"++++test error: ", s,newline,newline))
        END;

   REAL eps = 20.0*smallreal; #moan if discrepancy is larger than this#

   REAL sumdelta := 0, sumsqdelta := 0, maxdelta := 0, at,

   INT count := 0;

   PROC test = (REAL r)VOID:
        BEGIN
           REAL s = sqrt(r);
           REAL t = s*s;
           REAL d = ABS((r-t)/r);
           IF d>eps
           THEN print("sqrt("); print(r); print(") = "); print(s);
                warn("relative error in sqrt*sqrt exceeds 20*smallreal")
           ELSE REAL dd = d/smallreal;
                sumdelta +:= dd; sumsqdelta +:= dd*dd; count +:= 1;
                IF dd>maxdelta THEN maxdelta := dd; at := r FI
           FI
        END;

   REAL r := pi, REAL 1 = maxreal/4;
   WHILE test(r); r<1 DO r*:=pi OD;

   r := 1/pi;
   WHILE test(r); REAL s=r; r/:=pi;
          r+r<s AND r/=0
      DO SKIP OD;
```



```
   IF REAL r = sqrt(0); r/=0
   THEN print("sqrt(0)=");
           print(fixed(r,-(realwidth+1),realwidth-1));
        warn("sqrt(0) should be 0")
   FI;

   IF count/=0
   THEN print((newline,"Except when indicated above,",
              newline,"Maximum relative error=smallreal*"));
              print(fixed(maxdelta,-(realwidth%2+2),realwidth%2));
        print((newline,"Average relative error=smallreal*"));
              print(fixed(sumdelta/count,-(realwidth%2+2),realwidth%2)
        print((newline,"R.M.S. relative error =smallreal*"));
              print(fixed(sqrt(sumdelta/count),-(realwidth%2+2),
                                                    realwidth%2)
        print(newline)
   FI

END
```

- - . - -

```
#numr09#
BEGIN#Test exp#

   #N.B. This test should not be considered as certification of 'exp',
         but only as an indication that 'exp' has the right properties

   PROC warn = (STRING s)VOID:
        BEGIN
           print((newline,"++++test error: ", s,newline,newline))
        END;

   REAL minreal = 2/(smallreal*maxreal*smallreal);
      # must be close to the smallest real value > 0 ;
        this will probably work on most machines with normalized reals

   REAL e=exp(1);

   REAL y1,  #exp(x)#
        y2,  #exp(-x)#
        y3,  #exp(1/x)#
        y4;  #exp(-1/x)#

   REAL max1 := 0, sum1 := 0, sumsq1 := 0, INT c1 := 0, REAL at1;

   PROC test1 = (REAL x, y1, y2)VOID:
        # exp(x)*exp(-x)=1 #
        IF y1=0
        THEN print((newline,newline,"x=",x,newline,"exp(x)=0"))
```

```
     ELIF y2=0
     THEN print((newline,newline,"x=",-x,newline,"exp(x)=0"))
     ELIF REAL d = ABS(y1*y2 - 1)/smallreal;
        sum1 +:= d; sumsq1 +:= d*d; c1 +:= 1;
        max1<d
     THEN max1 := d; at1 := x
     FI;

REAL max2 := 0, sum2 := 0, sumsq2 := 0, INT c2 := 0, REAL at2;

PROC test2 = (REAL x, y)VOID:
     # exp(1+x)=e*exp(x) #
     IF y/=0 AND y<maxreal/3
     THEN IF REAL z = exp(1+x); z<=0
           THEN print((newline,
                      newline,"x=",1+x,
                      newline,"exp(x)=",z));
                warn("exp(x)<=0")
           ELIF REAL d = ABS((z - e*y)/z)/smallreal;
                sum2 +:= d; sumsq2 +:= d*d; c2 +:= 1;
                max2<d
           THEN max2 := d; at2 := x
           FI
     FI;

REAL max3 := 0, sum3 := 0, sumsq3 := 0, INT c3 := 0, REAL at3;

PROC test3 = (REAL x, y)VOID:
     # sqrt(exp(2x))=exp(x) #
     IF y/=0 AND y<sqrt(maxreal) AND y>sqrt(minreal)
     THEN IF REAL z = sqrt(exp(x+x)); z>0
           THEN REAL d = ABS((z - y)/y)/smallreal;
                sum3 +:= d; sumsq3 +:= d*d; c3 +:= 1;
                (max3<d| max3:=d; at3:=x)
           FI
     FI;

PROC test4 = (REAL x, y)VOID:
     # check 0<=x<1, 1+x<= exp(x) <= 1/(1-x) #
     IF 0<=x AND x<1
     THEN IF y<1+x
           THEN print((newline,
                      newline,"x=",x,
                      newline,"exp(x)= ",y,
                      newline,"1+x   = ",1+x));
                warn("exp(x) should exceed 1+x")
           ELIF y>1/(1-x)
           THEN print((newline,
                      newline,"x=",x,
                      newline,"exp(x)= ",y,
                      newline,"1/(1-x)=",1/(1-x)));
```



```
                warn("exp(x) should be less than 1/(1-x)")
           FI
     FI;

   REAL x := 1;
   WHILE x +:= random; REAL z = 1/x;
         y1 := exp(x); y2 := exp(-x); y3 := exp(z); y4 := exp(-z);
         test1(x,y1,y2); test1(z,y3,y4);
         test2(x,y1); test2(-x,y2); test2(z,y3); test2(-z,y4);
         test3(x,y1); test3(-x,y2); test3(z,y3); test3(-z,y4);
         test4(z,y3);
         y1<maxreal/3 AND y2>3*minreal
   DO SKIP OD;

   PROC p = (STRING s, REAL sum, sumsq, n, max, at)VOID:
        BEGIN
           print((newline,newline,s));
           print((newline,"Maximum relative error = smallreal*"));
             print(fixed(max,-(realwidth%2+2),realwidth%2));
           print((newline,"Occurred at x = ", at));
           print((newline,"Average relative error = smallreal*"));
             print(fixed(sum/n,-(realwidth%2+2),realwidth%2));
           print((newline,"R.M.S. relative error  = smallreal*"));
             print(fixed(sqrt(sumsq/n),-(realwidth%2+2),realwidth%2))
        END;

   p("Checks on exp(x)*exp(-x)=1", sum1, sumsq1, c1, max1, at1);
   p("Checks on exp(1+x)=exp(1)*exp(x)", sum2, sumsq2, c2, max2, at2)
   p("Checks on sqrt(exp(2*x))=exp(x)", sum3, sumsq3, c3, max3, at3);

   print(newline);
   print((newline,
     "e        = 2.71828182845904523536028747135266249775 72+ (Knuth)"
   print((newline,"exp(1)   = "));
     print(fixed(exp(1),-(real width+1),real width-1));
   print((newline,
     "e**-1    = 0.36787944117144232159552377016146086744 58+ (Knuth)"
   print((newline,"exp(-1)  = "));
     print(fixed(exp(-1),-(real width+1),real width-1));
   print((newline,
     "e**2     = 7.38905609893065022723042746057500781318 03+ (Knuth)"
   print((newline,"exp(2)   = "));
     print(fixed(exp(2),-(real width+1),real width-1));

   print((newline,"smallreal= "));
     print(fixed(small real,-2*real width,2*(realwidth-1)));
   print((newline,"         = ",small real))

END
```

- - . - -

```
#numr10#
BEGIN#Test ln#

   #N.B. This test should not be considered as certification of 'ln',
         but only as an indication that 'ln' has the right properties#

   PROC warn = (STRING s)VOID:
        BEGIN
           print((newline,newline,"+++test error: ", s,newline))
        END;

   REAL minreal = 2/(smallreal*maxreal*smallreal);
      # must be close to the smallest real value > 0 ;
        this will probably work on most machines with normalized reals #

   REAL e = exp(1);

   REAL y1,  #exp(x)#
        y2,  #exp(-x)#
        y3,  #exp(1/x)#
        y4;  #exp(-1/x)#
   REAL z1,  #ln(y1)#
        z2,  #ln(y2)#
        z3,  #ln(y3)#
        z4;  #ln(y4)#

   REAL max1 := 0, sum1 := 0, sumsq1 := 0, INT c1 := 0, REAL at1;

   PROC test1 = (REAL x, y, z)VOID:
        #ln(exp(x))=x#
        IF y>0.0
        THEN REAL d = ABS((x - z)/x)/smallreal;
             sum1 +:= d; sumsq1 +:= d*d; c1 +:= 1;
             (max1<d|max1 := d; at1 := x)
        FI;

   REAL max2 := 0, sum2 := 0, sumsq2 := 0, INT c2 := 0, REAL at2;

   PROC test2 = (REAL x, y)VOID:
        #ln(e*x)=1+ln(x)#
        IF x<maxreal/3
        THEN REAL z = ln(e*x);
             REAL z1 = y + 1;
             REAL d = ABS((z - z1)/z)/smallreal;
             sum2 +:= d; sumsq2 +:= d*d; c2 +:= 1;
             (max2<d|max2 := d; at2 := x)
        FI;

   REAL max3 := 0, sum3 := 0, sumsq3 := 0, INT c3 := 0, REAL at3;

   PROC test3 = (REAL x, y)VOID:
```



```
        #2*ln(sqrt(x))=ln(x)#
        IF REAL z = ln(sqrt(x)); REAL z1 = z + z;
            REAL d = ABS((y - z1)/y)/smallreal;
            sum3 +:= d; sumsq3 +:= d*d; c3 +:= 1;
            max3<d
        THEN max3 := d; at3 := x
        FI;

   REAL max4 := 0, sum4 := 0, sumsq4 := 0, INT c4 := 0, REAL at4;

   PROC test4 = (REAL x, y, z)VOID:
        #ln(1/x)=-ln(x)#
        IF REAL z1 = ABS y + ABS z;
            z1 /= 0
        THEN REAL d = ABS(2*(y+z)/z1)/smallreal;
             sum4 +:= d; sumsq4 +:= d*d; c4 +:= 1;
             (max4<d|max4 := d; at4 := x)
        FI;

   PROC test5 = (REAL x, y)VOID:
        #x>0, 1-1/x <= ln(x) <= x-1#
        IF y<1.0-1.0/x
        THEN print((newline,
                   newline,"x=",x,
                   newline,"ln(x) = ",y,
                   newline,"1-1/x = ",1-1/x));
             warn("ln(x) should not be less than 1-1/x")
        ELIF y>x-1
        THEN print((newline,
                   newline,"x=",x,
                   newline,"ln(x) = ",y,
                   newline,"x-1   =",x-1));
             warn("ln(x) should not exceed x-1")
        FI;

   REAL x := 1;
   WHILE x +:= random; REAL x2 = -x, x3 = 1/x; REAL x4 = -x3;
         y1 := exp(x); y2 := exp(x2); y3 := exp(x3); y4 := exp(x4);
         z1 := ln(y1); z2 := ln(y2); z3 := ln(y3); z4 := ln(y4);
         test1(x,y1,z1); test1(x2,y2,z2); test1(x3,y3,z3);
                                                    test1(x4,y4,z
         test2(y1,z1); test2(y2,z2); test2(y3,z3); test2(y4,z4);
         test3(y1,z1); test3(y2,z2); test3(y3,z3); test3(y4,z4);
         test4(y1,z1,z2); test4(y3,z3,z4);
         test5(y1,z1); test5(y2,z2); test5(y3,z3); test5(y4,z4);
         y1<maxreal/3 AND y2>3*minreal
   DO SKIP OD;

   PROC p = (STRING s, REAL sum, sumsq, n, max, at)VOID:
        BEGIN
           print((newline,newline,s));
```

```
        print((newline,"Maximum relative error = smallreal*"));
         print(fixed(max,-(realwidth%2+2),realwidth%2));
        print((newline,"Occurred at x = ", at));
        print((newline,"Average relative error = smallreal*"));
         print(fixed(sum/n,-(realwidth%2+2),realwidth%2));
        print((newline,"R.M.S. relative error  = smallreal*"));
         print(fixed(sqrt(sumsq/n),-(realwidth%2+2),realwidth%2))
      END;

  p("Checks on ln(exp(x))=x", sum1, sumsq1, c1, max1, at1);
  p("Checks on ln(e*x)=1+ln(x)", sum2, sumsq2, c2, max2, at2);
  p("Checks on 2*ln(sqrt(x))=ln(x)", sum3, sumsq3, c3, max3, at3);
  p("Checks on ln(1/x)=-ln(x)", sum4, sumsq4, c4, max4, at4);

  print(newline);
  print((newline,"log 1      = 0"));
  print((newline,"ln(1)      = "));
  print(fixed(ln(1),-(realwidth+1),realwidth-1));
  print((newline,
   "log 2      = 0.693147180559945309417232121458176568075 5+ (Knuth)"));
  print((newline,"ln(2)      = "));
  print(fixed(ln(2),-(realwidth+1),realwidth-1));
  print((newline,
   "log 3      = 1.098612288668109691395245236922525704647 5- (Knuth)"));
  print((newline,"ln(3)      = "));
  print(fixed(ln(3),-(realwidth+1),realwidth-1));
  print((newline,
   "log 10     = 2.302585092994045684017991454684364207601 1+ (Knuth)"));
  print((newline,"ln(10)     = "));
  print(fixed(ln(10),-(realwidth+1),realwidth-1));
  print((newline,
   "-loglog 2 = 0.366512920581664327012439158232669469454 3- (Knuth)"));
  print((newline,"-ln(ln(2))= "));
  print(fixed(-ln(ln(2)),-(realwidth+1),realwidth-1));

  print((newline,"smallreal = "));
  print(fixed(small real,-2*realwidth,2*(realwidth-1)));
  print((newline,"          = ",small real,newline))

ND
```

- - . - -

```
umr11#
EGIN  #Test trig functions 1#

  #N.B. This test should not be considered as certification of trig
        functions, but only as an indication that trig functions have
        the right properties#
```



```
# Value checks #

REAL eps = 10.0*smallreal;

PROC warn = (STRING s)VOID:
     BEGIN
        print((newline,"++++test error: ", s,newline))
     END;

REAL zero = 0, half = 0.5, one = 1, two = 2, three = 3,
     four = 4, five = 5, six = 6, seven = 7, eight = 8,
     twelve = 12, sixteen = 16, thirtytwo = 32;

[]REAL theta = []REAL(zero,pi/two,pi,three*pi/two,pi+pi,
                                        five*pi/two,three*pi)[:AT 0],
       phi   = []REAL(zero,pi/six,pi/four,pi/three,pi/two)[:AT 0],
       sphi  = []REAL(zero,half,sqrt(half),sqrt(0.75),one)[:AT 0],
       tphi  = []REAL(zero,sqrt(one/three),one,sqrt(three),
                                                 maxreal)[:AT 0],

[]STRING angle = []STRING("0","pi/6","pi/4","pi/3","pi/2")[:AT 0];
INT upb = UPB phi;

PROC test = (REAL a, INT i, j, REAL s, c, t)VOID:
     BEGIN
        PROC printangle = VOID:
             BEGIN
                IF a<zero THEN print("-(") FI;
                IF ODD j
                THEN IF j=1
                     THEN print("pi/2")
                     ELSE print(whole(j,0)); print("*pi/2")
                     FI
                ELSE IF j/=0
                     THEN IF j=1
                          THEN print("pi")
                          ELSE print(whole(j%2,0)); print("*pi")
                          FI
                     FI
                FI;
                IF i=0
                THEN IF j=0 THEN print("0") FI
                ELSE IF j/=0 THEN print("+") FI;
                     print(angle[i])
                FI;
                IF a<0 THEN print(")") FI
             END;

        IF ABS(sin(a) - s)>eps
        THEN print((newline,
                  newline,"sin(")); printangle;
```

```
                                        print((") =", sin(a)));
        print((newline,"expected =",s));
        warn("probable error in ''sin''")
    FI;

    IF ABS(cos(a) - c)>eps
    THEN print((newline,
              newline,"cos(")); printangle;
                                        print((") =", cos(a)));
        print((newline,"expected =",c));
        warn("probable error in ''cos''")
    FI;

    IF(ODD j | i/=0 | i/=upb)
    THEN IF t=maxreal THEN SKIP
        ELIF ABS(tan(a) - t)>two*eps
        THEN print((newline,
                  newline,"tan(")); printangle;
                                        print((") =", tan(a)));
            print((newline,"expected =",t));
            warn("probable error in ''tan''")
        FI
    FI
  END;

FOR j FROM 0 TO UPB theta
  DO FOR i FROM 0 TO upb-1
      DO REAL a = theta[j] + phi[i];
         REAL s1 = sphi[(ODD j | upb - i | i)],
              c1 = sphi[(ODD j | i | upb - i)],
              t = (ODD j | -tphi[upb - i] | tphi[i]);
         REAL s = (ODD(j%2) | -s1 | s1),
              c = (ODD((j+1)%2) | -c1 | c1);
         test(a,i,j,s,c,t);
         test(-a,i,j,-s,c,-t)
      OD
  OD;

#Check identities:
      sin(x) = 2*tan(x/2)/(1+tan(x/2)**2),
      cos(x) = (1-tan(x/2)**2)/(1+tan(x/2)**2),
      tan(x) = 2*tan(x/2)/(1-tan(x/2)**2).#

REAL sums := zero, sumc := zero, sumt := zero,
     sumsqs := zero, sumsqc := zero, sumsqt := zero,
     maxs := zero, maxc := zero, maxt := zero,
     ats, atc, att,
INT cs := 0, cc := 0, ct := 0;

TO 200
DO REAL a = random; REAL aby2 = a/2; REAL tanaby2 = tan(aby2);
```



```
   REAL tanaby2sq = tanaby2*tanaby2;
   REAL snum = tanaby2+tanaby2, cnum = 1 - tanaby2sq,
        denom = 1 + tanaby2sq;
   REAL s = snum/denom, c = cnum/denom,
        t = (cnum<2/maxreal | -1 | snum/cnum);

   IF REAL sina = sin(a);
      REAL d1 = 2*ABS(sina - s), d2 = ABS sina + ABS s;
      d2/=zero
   THEN REAL d = (d1/d2)/smallreal;
        sums +:= d; sumsqs +:= d*d; cs +:= 1;
        (d>maxs| maxs := d; ats := a)
   FI;

   IF REAL cosa = cos(a);
      REAL d1 = 2.0*ABS(cosa - c), d2 = ABS cosa + ABS c;
      d2/=zero
   THEN REAL d = (d1/d2)/smallreal;
        sumc +:= d; sumsqc +:= d*d; cc +:= 1;
        (d>maxc| maxc := d; atc := a)
   FI;

   IF t>=zero
   THEN REAL tana = tan(a);
        REAL d1 = 2*ABS(tana - t), d2 = ABS tana + ABS t;
        IF d2/=zero
        THEN REAL d = (d1/d2)/smallreal;
             sumt +:= d; sumsqt +:= d*d; ct +:= 1;
             (d>maxt| maxt := d; att := a)
        FI
   FI
OD;

PROC p = (STRING s, REAL max,at,sum,sumsq, INT c) VOID:
  (print((newline,newline,s));
   print((newline,"Max. relative error = smallreal*"));
   print(fixed(max,-(realwidth%2+1),realwidth%2-1));
   (max/=zero|print((newline,"Occurred at x = ",at)));
   print((newline,"Average relative error = smallreal*"));
   print(fixed(sum/c,-(realwidth%2+1),realwidth%2-1));
   print((newline,"R.M.S. relative error  = smallreal*"));
   print(fixed(sqrt(sumsq/c),-(realwidth%2+1),realwidth%2-1)));

p("Checks on sin(a)=2*tan(a/2)/(1+tan(a/2)*2):",
   maxs, ats, sums, sumsqs, cs);
p("Checks on cos(a)=(1-tan(a/2)*2)/(1+tan(a/2)*2):",
   maxc, atc, sumc, sumsqc, cc);
p("Checks on tan(a)=2*tan(a/2)/(1-tan(a/2)*2):",
   maxt, att, sumt, sumsqt, ct)
END
```

- - . - -

```
#numr12#
BEGIN  #Test trig functions 2#

   #N.B. This test should not be considered as certification of trig
         functions, but only as an indication that trig functions have
         the right properties#

   #Spot checks#

   REAL zero = 0, half = 0.5, one = 1, two = 2, three = 3,
        four = 4, five = 5, six = 6, seven = 7, eight = 8,
        twelve = 12.0, sixteen = 16.0, thirtytwo = 32.0;

   print((newline,"Spot checks:",newline));

   FOR i TO 12
   DO STRUCT(STRING s1,s2,REAL s,STRING s3,REAL c)z =
      CASE i
      IN
       ("sin 0           = zero",
          "sin(0)          = ",zero,
          "cos(pi/2)       = ",pi/two),
       ("sin pi/24       = 0.13052619222005159154840622789548901 (Hart)",
          "sin(pi/24)      = ",pi/24,
          "cos(11*pi/24)   = ",11.0*pi/24.0),
       ("sin pi/16       = 0.19509032201612826784828486847702224 (Hart)",
          "sin(pi/16)      = ",pi/sixteen,
          "cos(7*pi/16)    = ",seven*pi/sixteen),
       ("sin 1/4         = 0.24740395925452292959684870484938920 (Hart)",
          "sin(1/4)        = ",0.25,
          "cos((2*pi-1)/4)= ",(pi+pi-1)/four),
       ("sin pi/12       = 0.25881904510252076234889883762404832 (Hart)",
          "sin(pi/12)      = ",pi/twelve,
          "cos(5*pi/12)    = ",five*pi/twelve),
       ("sin 1/2         = 0.47942553860420300027328793521557139 (Hart)",
          "sin(1/2)        = ",half,
          "cos((pi-1)/2)   = ",(pi-one)/two),
       ("sin pi/6        = 0.5",
          "sin(pi/6)       = ",pi/six,
          "cos(pi/3)       = ",pi/three),
       ("sin pi/4        = 0.70710678118654752440084436210484903 (Hart)",
          "sin(pi/4)       = ",pi/four,
          "cos(pi/4)       = ",pi/four),
       ("sin 1           = 0.84147098480789650665250232163029900 (Hart)",
          "sin(1)          = ",one,
          "cos(pi/2-1)     = ",(pi-two)/two),
       ("sin pi/3        = 0.86602540378443864676372317075293618 (Hart)",
          "sin(pi/3)       = ",pi/three,
```



```
          "cos(pi/6)       = ",pi/six),
       ("sin 3*pi/8      = 0.92387953251128675612818319839678828 (Hart
          "sin(3*pi/8)     = ",three*pi/eight,
          "cos(pi/8)       = ",pi/eight),
       ("sin 5*pi/12     = 0.96592582628906828674974319972889736 (Hart
          "sin(5*pi/12)    = ",five*pi/twelve,
          "cos(pi/12)      = ",pi/twelve),
         SKIP
      ESAC;

      print((newline,s1 OF z,
            newline,s2 OF z));
         print(fixed(sin(s OF z),-(realwidth+1),realwidth-1));
      print((newline,s3 OF z));
         print(fixed(cos(c OF z),-(realwidth+1),realwidth-1));
      print(newline)

   OD;


   FOR i TO 23
   DO STRUCT(STRING s1,s2,REAL t)z =
      CASE i
      IN
      ("tan 0           = 0",
          "tan(0)          = ",zero),
      ("tan pi/32       = 0.09849140335716425307719752129132743 (Hart)
          "tan(pi/32)      = ",pi/thirtytwo),
      ("tan pi/16       = 0.19891236737965800691159762264467622 (Hart)
          "tan(pi/16)      = ",pi/sixteen),
      ("tan 1/4         = 0.25534192122103626650448223649047368 (Hart)
          "tan(1/4)        = ",0.25),
      ("tan pi/12       = 0.26794919243112270647255365849412763 (Hart)
          "tan(pi/12)      = ",pi/twelve),
      ("tan 3*pi/32     = 0.30334668360734239167588394694129987 (Hart)
          "tan(3*pi/32)    = ",three*pi/thirtytwo),
      ("tan pi/8        = 0.41421356237309504880168872420969807 (Hart)
          "tan(pi/8)       = ",pi/eight),
      ("tan 5*pi/32     = 0.53451113595079164108968596129536290 (Hart)
          "tan(5*pi/32)    = ",five*pi/thirtytwo),
      ("tan 1/2         = 0.54630248984379051325517946578028538 (Hart)
          "tan(1/2)        = ",half),
      ("tan pi/6        = 0.57735026918962576450914878050195745 (Hart)
          "tan(pi/6)       = ",pi/six),
      ("tan 3*pi/16     = 0.66817863791929891999775768652308076 (Hart)
          "tan(3*pi/16)    = ",three*pi/sixteen),
      ("tan 7*pi/32     = 0.82067879082866033097228198533101159 (Hart)
          "tan(7*pi/32)    = ",seven*pi/thirtytwo),
      ("tan pi/4        = 1.0",
          "tan(pi/4)       = ",pi/four),
      ("tan 9*pi/32     = 1.21850362558797634479547723062036405 (Hart)
```

```
        "tan(9*pi/32)   = ",9.0*pi/thirtytwo),
      ("tan 5*pi/16     = 1.4966057626654890176011351349424769l (Hart)",
        "tan(5*pi/16)   = ",five*pi/sixteen),
      ("tan 1           = 1.557407724654902230506974807458360l7 (Hart)",
        "tan(1)         = ",one),
      ("tan pi/3        = 1.732050807568877293527446341505872l6 (Hart)",
        "tan(pi/3)      = ",pi/three),
      ("tan 11*pi/32    = 1.870868411789389481085201334341524l3 (Hart)",
        "tan(11*pi/32)  = ",11.0*pi/thirtytwo),
      ("tan 3*pi/8      = 2.414213562373095048801688724209698l7 (Hart)",
        "tan(3*pi/8)    = ",three*pi/eight),
      ("tan 13*pi/32    = 3.296558208938320426878154216826253l0 (Hart)",
        "tan(13*pi/32)  = ",13*pi/thirtytwo),
      ("tan 5*pi/12     = 3.732050807568877293527446341505872l6 (Hart)",
        "tan(5*pi/12)   = ",five*pi/twelve),
      ("tan 7*pi/16     = 5.027339492125848104514975071064072l8 (Hart)",
        "tan(7*pi/16)   = ",seven*pi/sixteen),
      ("tan 15*pi/32    = 10.153170387608860462107147663419472l0 (Hart)",
        "tan(15*pi/32)  = ",15*pi/thirtytwo),
       SKIP
  ESAC;

    print((newline,s1 OF z,
          newline,s2 OF z));
       print(fixed(tan(t OF z),-(realwidth+1),realwidth-1));
    print(newline)

  OD;

  print((newline,"smallreal      = "));
       print(fixed(smallreal,-2*realwidth,2*(realwidth-1)));
  print((newline,"               =",smallreal));
  print(newline)

END
```

- - . - -

```
#numr13#
BEGIN  #Test inverse trig functions#

  PROC warn = (STRING s)VOID:
       BEGIN
          print((newline,s))
       END;

  REAL zero = 0;
  REAL sums1:= zero, sumsqs1:= zero, maxs1:= zero, ats1, INT cs1:= 0;
  REAL sums2:= zero, sumsqs2:= zero, maxs2:= zero, ats2, INT cs2:= 0;
  REAL sums3:= zero, sumsqs3:= zero, maxs3:= zero, ats3, INT cs3:= 0;
```



```
  REAL sumc1:= zero, sumsqc1:= zero, maxc1:= zero, atc1, INT cc1:= 0
  REAL sumc2:= zero, sumsqc2:= zero, maxc2:= zero, atc2, INT cc2:= 0
  REAL sumc3:= zero, sumsqc3:= zero, maxc3:= zero, atc3, INT cc3:= 0
  REAL sumt1:= zero, sumsqt1:= zero, maxt1:= zero, att1, INT ct1:= 0
  REAL sumt2:= zero, sumsqt2:= zero, maxt2:= zero, att2, INT ct2:= 0
  REAL sumt3:= zero, sumsqt3:= zero, maxt3:= zero, att3, INT ct3:= 0
  REAL sumsc:= zero, sumsqsc:= zero, maxsc:= zero, atsc, INT csc:= 0
  REAL sumcs:= zero, sumsqcs:= zero, maxcs:= zero, atcs, INT ccs:= 0

  REAL piby2 = pi/2;

  PROC asin = (REAL x, PROC VOID 1)REAL:
       IF# x positive, 0<=arcsin(x)<pi/2 #
          REAL y = arcsin(x);
          PROC e = (STRING s)VOID:
               (print((newline,
                     newline,"arcsin(",x," ) =",y));
                warn(s);
                1);
          y<zero THEN e("arcsin yields result of wrong sign"); SKIP
       ELIF y>piby2 THEN e("arcsin exceeds pi/2");
                         #allow rounding up#
                         SKIP

       ELSE IF# sin(arcsin(x))=x ? #
               REAL z = sin(y);
               REAL d1 = ABS z + ABS x;
               d1 = zero
            THEN cs1 +:= 1
            ELIF REAL d2 = ABS((ABS z - ABS x)/smallreal);
                 REAL d = (d2+d2)/d1;
                 sums1 +:= d; sumsqs1 +:= d*d; cs1 +:= 1;
                 d>maxs1
            THEN maxs1 := d; ats1 := x
            FI;

            IF# arcsin(-x)=-arcsin(x) ? #
               REAL z = arcsin(-x);
               SIGN z /= -SIGN y
            THEN print((newline,
                      newline,"arcsin(",-x," ) =",z));
                 warn("arcsin yields result of wrong sign");
                 1
            ELIF REAL d1 = ABS y + ABS z;
                 d1=zero
            THEN cs2 +:= 1
            ELIF REAL d2 = ABS((ABS y - ABS z)/smallreal);
                 REAL d = (d2+d2)/d1;
                 sums2 +:= d; sumsqs2 +:= d*d; cs2 +:= 1;
                 d>maxs2
            THEN maxs2 := d; ats2 := x
```

```
        FI;

        y
    FI;

PROC acos = (REAL x, PROC VOID l)REAL:
    IF# x positive, 0<=arccos(x)<pi/2 #
       REAL y = arccos(x);
       PROC e = (STRING s)VOID:
            (print((newline,
                  newline,"arccos(",x," ) = ",y));
             warn(s);
             l);
       y<zero THEN e("arccos yields result of wrong sign"); SKIP
    ELIF y>piby2 THEN e("arccos exceeds pi/2");
                      #allow rounding up#
                      SKIP

    ELSE IF x>=smallreal
          #otherwise arccos(x)=pi/2 and cos(arccos(x))=0#
       THEN IF# cos(arccos(x))=x ? #
                 REAL z = cos(y);
                 REAL dl = ABS z + ABS x;
                 dl = zero
            THEN ccl +:= l
            ELSE REAL d2 = ABS((ABS z - ABS x)/smallreal);
                 REAL d = (d2+d2)/dl;
                 sumcl +:= d; sumsqcl +:= d*d; ccl +:= l;
                 (d>maxcl| maxcl := d; atcl := x)
            FI
       FI;

       IF# arccos(-x)=pi-arccos(x) ? #
          REAL z = arccos(-x);
          z<piby2
       THEN print((newline,
                 newline,"arccos(",-x," ) =",z));
            warn("result should exceed pi/2");
            l
       ELIF z>pi
       THEN print((newline,
                 newline,"arccos(",-x," ) =",z));
            warn("arccos should not exceed pi");
            l
       ELIF REAL zz = pi-y;
            REAL dl = ABS zz + ABS z;
            dl=zero
       THEN cc2 +:= l
       ELIF REAL d2 = ABS((ABS zz - ABS z)/smallreal);
            REAL d = (d2+d2)/dl;
            sumc2 +:= d; sumsqc2 +:= d*d; cc2 +:= l;
```



```
                 d>maxc2
            THEN maxc2 := d; atc2 := x
            FI;

            y
       FI;

  PROC atan = (REAL x, PROC VOID l)REAL:
       IF# x positive, 0<=arctan(x)<pi/2 #
          REAL y = arctan(x);
          PROC e = (STRING s)VOID:
               (print((newline,
                     newline,"arctan(",x," ) =",y));
                warn(s);
                l);
          y<zero THEN e("arctan yields result of wrong sign"); SKIP
       ELIF y>piby2 THEN e("arctan exceeds pi/2");
                         #allow rounding up#
                         SKIP

       ELIF# tan(arctan(x))=x ? #
                x<maxreal/2
       THEN IF REAL z = tan(y);
               REAL dl = ABS z + ABS x;
               dl = zero
            THEN ctl +:= l
            ELIF REAL d2 = ABS((ABS z - ABS x)/smallreal);
                 REAL d = (d2+d2)/dl;
                 sumtl +:= d; sumsqtl +:= d*d; ctl +:= l;
                 d>maxtl
            THEN maxtl := d; attl := x
            FI;

            IF# arctan(-x)=-arctan(x) ? #
               REAL z = arctan(-x);
               SIGN z /= -SIGN y
            THEN print((newline,
                      newline,"arctan(",-x," ) =",z));
                 warn("arctan yields result of wrong sign");
                 l
            ELIF REAL dl = ABS y + ABS z;
                 dl=zero
            THEN ct2 +:= l
            ELIF REAL d2 = ABS((ABS y - ABS z)/smallreal);
                 REAL d = (d2+d2)/dl;
                 sumt2 +:= d; sumsqt2 +:= d*d; ct2 +:= l;
                 d>maxt2
            THEN maxt2 := d; att2 := x
            FI;

            y
```

```
     FI;

PROC test = (REAL a, b, h)VOID:
     BEGIN
        REAL arcsin x = asin(a/h,VOID: GOTO l),
             arcsin y = asin(b/h,VOID: GOTO l),
             arccos x = acos(a/h,VOID: GOTO l),
             arccos y = acos(b/h,VOID: GOTO l),
             arctan x = atan(a/b,VOID: GOTO l),
             arctan y = atan(b/a,VOID: GOTO l);

        IF# arcsin(x)+arcsin(y)=pi/2 #
           REAL z = arcsin x + arcsin y;
           REAL d = ABS(z-piby2)/(piby2*smallreal);
           sums3 +:= d; sumsqs3 +:= d*d; cs3 +:= l;
           d>maxs3
        THEN maxs3 := d; ats3 := a/h
        FI;

        IF# arccos(x)+arccos(y)=pi/2 #
           REAL z = arccos x + arccos y;
           REAL d = ABS(z-piby2)/(piby2*smallreal);
           sumc3 +:= d; sumsqc3 +:= d*d; cc3 +:= l;
           d>maxc3
        THEN maxc3 := d; atc3 := b/h
        FI;

        IF# arctan(x)+arctan(y)=pi/2 #
           REAL z = arctan x + arctan y;
           REAL d = ABS(z-piby2)/(piby2*smallreal);
           sumt3 +:= d; sumsqt3 +:= d*d; ct3 +:= l;
           d>maxt3
        THEN maxt3 := d; att3 := a/b
        FI;

        IF#arcsin(x)=arccos(y)#
           REAL z = arcsin x + arccos y;
           z=zero THEN csc +:= l
        ELIF REAL zz = ABS(arcsin x - arccos y)/z;
             REAL d = (zz + zz)/smallreal;
             sumsc +:= d; sumsqsc +:= d*d; csc +:= l;
             d>maxsc
        THEN maxsc := d; atsc := a/h
        FI;

        IF#arccos(x)=arcsin(y)#
           REAL z = arccos x + arcsin y;
           z=zero THEN ccs +:= l
        ELIF REAL zz = ABS(arccos x - arcsin y)/z;
             REAL d = (zz + zz)/smallreal;
             sumcs +:= d; sumsqcs +:= d*d; ccs +:= l;
```



```
             d>maxcs
        THEN maxcs := d; atcs := b/h
        FI;

         l: SKIP
        END;

     PROC gcd = (INT a, b)INT:
          IF INT c = a MOD b; c=0
          THEN b
          ELSE gcd(b,c)
          FI;

     INT c := 200;  #number of triangles tested#
     FOR i FROM 2 TO maxint WHILE c>0
       DO FOR j FROM i-l BY-2 TO l
            WHILE#generate Pythagorean triangle#
                 IF gcd(i,j)=l AND(ODD i /= ODD j)
                 THEN REAL i2 = i*i, j2=j*j, ij = i*j;
                      REAL short = i2 - j2, long = ij + ij,
                                                        hypot = i2 + j2;
                      test(short,long,hypot);
                      c-:=l
                 ELSE c
                 FI>0
             DO SKIP OD
       OD;

     PROC p = (STRING s, REAL max, at, sum, sumsq, INT c) VOID:
        (print((newline,newline,s,newline));
         print(("Maximum relative error = smallreal*",
                fixed(max,-(realwidth%2+2),realwidth%2),newline));
         (max/=zero|print(("Occurred at x = ",at,newline)));
         print(("Average relative error = smallreal*",
                fixed(sum/c,-(realwidth%2+2),realwidth%2),newline));
         print(("R.M.S. relative error = smallreal*",
                fixed(sumsq/c,-(realwidth%2+2),realwidth%2),newline));
         print(("Number of tests = ",whole(c,-5))));

     p("Checks on sin(arcsin(x))=x :",
        maxsl,atsl,sumsl,sumsqsl,csl);
     p("Checks on arcsin(-x)=-arcsin(x) :",
        maxs2,ats2,sums2,sumsqs2,cs2);
     p("Checks on arcsin(x)+arcsin(y)=pi/2 :",
        maxs3,ats3,sums3,sumsqs3,cs3);
     p("Checks on cos(arccos(x))=x :",
        maxcl,atcl,sumcl,sumsqcl,ccl);
     p("Checks on arccos(-x)=pi-arccos(x) :",
        maxc2,atc2,sumc2,sumsqc2,cc2);
     p("Checks on arccos(x)+arccos(y)=pi/2 :",
        maxc3,atc3,sumc3,sumsqc3,cc3);
```

```
p("Checks on tan(arctan(x))=x :",
   maxt1,att1,sumt1,sumsqt1,ct1);
p("Checks on arctan(-x)=-arctan(x) :",
   maxt2,att2,sumt2,sumsqt2,ct2);
p("Checks on arctan(x)+arctan(y)=pi/2 :",
   maxt3,att3,sumt3,sumsqt3,ct3);
p("Checks on arcsin(x)=arccos(y) :",
   maxsc,atsc,sumsc,sumsqsc,csc);
p("Checks on arccos(x)=arcsin(y) :",
   maxcs,atcs,sumcs,sumsqcs,ccs);

#Special values#

IF REAL a = asin(zero,VOID: GOTO l1); a/=zero
THEN print((newline,newline,"arcsin(0) =",a));
     warn("arcsin(0) should be 0")
FI;
l1:
IF REAL a = asin(1,VOID: GOTO l2); a/=piby2
THEN print((newline,newline,
        "arcsin(1) differs from pi/2 by smallreal*",
        fixed(ABS(piby2-a)/smallreal,-(realwidth%2+2),realwidth%2)))
FI;
l2:
IF REAL a = acos(1,VOID: GOTO l3); a/=zero
THEN print((newline,newline,"arccos(1) =",a));
     warn("arccos(1) should be 0")
FI;
l3:
IF REAL a = acos(zero,VOID: GOTO l4); a/=piby2
THEN print((newline,newline,
        "arccos(0) differs from pi/2 by smallreal*",
        fixed(ABS(piby2-a)/smallreal,-(realwidth%2+2),realwidth%2)))
FI;
l4:
IF REAL a = arccos(-1); a/=pi
THEN print((newline,newline,
        "arccos(-1) differs from pi by smallreal*",
        fixed(ABS(pi-a)/smallreal,-(realwidth%2+2),realwidth%2)))
FI;
l5:
IF REAL a = atan(zero,VOID: GOTO l6); a/=zero
THEN print((newline,newline,"arctan(0) =",a));
     warn("arctan(0) should be 0")
FI;
l6:
print((newline,newline,"smallreal =",smallreal,newline))
ID
```

- - . - -


```
#appl01#
BEGIN # ALGOL 68 program, TJD 730705.
    Calculates all increasing sequences
    adding up to a given integer from 1 to 10 #

    [1 : 4] INT a;

    PROC print solution = (INT p) VOID:
    print( (a[1:p], newline) );

    PROC build up = (INT p, rest) VOID:
    IF rest = 0 THEN print solution(p)
    ELSE
        FOR k FROM (p=0|1|a[p]+1) TO rest DO
        (a[p+1]:= k; build up(p+1, rest-k))OD
    FI;

    FOR g TO 10 DO
     print((newline, g, " =", newline)); build up(0, g)  OD

    # For an ALGOL 60 program yielding the same output
    see Th.J. Dekker, Syllabus Informatica,
    Instituut voor Toepassingen van de Wiskunde,
    Universiteit van Amsterdam, 1972, page 81 - 82 #
END
```

- - . - -

```
#appl02#
BEGIN # ALGOL 68 program, TJD 730706.
    Calculates all increasing sequences adding up
    to a given integer from 1 to 10.
    See ALGOL 68 program TJD 730705 #

    MODE LIST = STRUCT(INT summand, REF LIST link);

    HEAP LIST zero:= (0, NIL);

    PROC print solution = VOID:
        print((straighten(link OF zero), newline));

    PROC straighten = (REF LIST l) [] INT :
    IF l :=: NIL THEN   (     # empty #   )   ELSE
        [] INT st = straighten(link OF l);
        [ 0 : UPB st ] INT r;
        r[0]:= summand OF l; r[ 1 : UPB st ]:= st;
        r[ @ 1 ]
    FI #straighten# ;

    PROC build = (REF LIST p, INT rest) VOID:
```

```
    IF rest = 0 THEN print solution
    ELSE FOR k FROM summand OF p + 1 TO rest DO
        (HEAP LIST q:= (k, NIL);
            link OF p:= q;
            build(q, rest-k) ) OD
    FI;

    FOR g TO 10 DO
     print((newline, g, " =", newline)); build(zero, g) OD
END
```

- - . - -

```
#app103#
BEGIN # ALGOL 68 program TJD 730701.
    This program prints a difference table
    of a 4-th degree polynomial. #

    [0:5] INT a;

    OP MIM = (INT a, b) INT: (a  <= b | a | b );
    PRIO MIM = 1;

    PROC pol4 = (INT x) INT: x * (x + 1) * (x + 2) * (x + 3);

    FOR n FROM 0 TO 20 DO
     INT kmax = n MIM 5;
        [0: kmax] INT b;
        b[0]:= pol4(n);
        FOR k TO kmax DO b[k]:= b[k-1] - a[k-1] OD;
        a[0 : kmax AT 0]:= b;
        print((n, b, newline))
    OD
END
```

- - . - -

```
#app104#
BEGIN # 1. Sets in ALGOL 68; 2. Pebble problem of E.W. Dijkstra #

    MODE RED =   REF STRUCT(RED red),
         WHITE = REF STRUCT(WHITE white),
         BLUE =  REF STRUCT(BLUE blue);

    MODE STONE = UNION (RED, WHITE, BLUE);

    PROC sort = (REF [] STONE st) VOID:
    (INT pr:= 1, pw:= 1, pb:= UPB st;
```



```
        PRIO EXCH = 1;
        OP EXCH = (REF STONE a, b) VOID:
        (STONE c = b; b:= a; a:= c);

        TO UPB st
        DO CASE st[pw]
           IN(RED):   (st[pr] EXCH st[pw]; pr+:= 1; pw+:= 1),
             (WHITE): pw+:= 1,
             (BLUE):  (st[pw] EXCH st[pb]; pb -:= 1)
           ESAC
        OD
    );

    OP PRINT = (REF [] STONE st) VOID:
    ( print(newline);
        FOR i TO UPB st
        DO print((st[i] | (RED):"r", (WHITE):"w", (BLUE):"b"))
        OD
    );

    INT n = 20;
    [ 1 : n ] STONE stone;
    FOR i TO UPB stone DO stone[i]:=
        (ENTIER (random * 3) + 1 |
            RED(NIL), WHITE(NIL), BLUE(NIL) ) OD;
    PRINT stone; sort(stone); PRINT stone
END
```

- - . - -

```
#app105#
BEGIN # Collateral sorting #

  PROC quicksort = (REF [] ITEM a) VOID:
    # quicksort requires the operator < to be defined for two ITEM's
  IF INT m = LWB a, n = UPB a; m < n THEN
    STRUCT(INT left, right) l =
        # 'l' is a border running from 'left' to 'right' such that:
          1. all elements left of the border are smaller than those
              right of the border,
          2. the border contains at least one element.
        #
      BEGIN INT f = # random # ENTIER((n - m + 1) * random + m);
        ITEM x = a[f];

        PROC swap = (REF ITEM a,b) VOID:
        BEGIN ITEM h=a; a:=b; b:=h END;

        INT i:= m, j := n;
              # a[m-1] < a[f] < a[n+1] #
```

```
      split:
        FOR k FROM i BY 1 TO n DO
        IF x<a[k] THEN i:= k; end_left FI OD;
        i:= n + 1;
      end_left:
            # a[f] < a[i] -> i /= f #

        FOR k FROM j BY -1 TO m DO
        IF a[k]<x THEN j:= k; end_right FI OD;
        j:= m - 1;
      end_right:
            # a[j] < a[f] -> f /= j #
            # a[j] < a[i] -> i /= j #

        IF i<j THEN swap(a[i],a[j]); i+:=1; j-:=1; split
            # i => j, i /= j -> i > j -> i - j > 0 #
        ELIF i<f THEN swap(a[i],a[f]); i+:=1 # i - j > 1 #
            # i >= f, i /= f -> i > f #
        ELIF f<j THEN swap(a[f],a[j]); j-:=1 # i - j > 1 #
            # f >= j, f /= j -> f > j; i>f,j>f -> i>f>j -> #
            # i-j > f-j > 0 -> i-j > 0 #
        FI;

        (j, i) # i - j > 1 #
    END;

  (quicksort(a[ : left OF l]), quicksort(a[right OF l : ]))
FI;

MODE ITEM = REAL;

PROC test = (INT max) VOID:
BEGIN [ 1 : max ] REAL a;
  FOR i TO max DO a[i]:= random OD;
  REAL time:= clock;
  quicksort(a); time:= clock - time;
  print(("Sorted", max, " numbers, time taken", time, " sec., i. e.,",
         time / (max * ln(max) / ln(2)), " per n ln n.", newline));

  FOR i TO max - 1
  DO IF a[i] > a[i+1] THEN print("Error ") FI OD
END # test # ;

test(100); test(1 000); test(10 000)
ND
```

- - . - -


```
#app106#
# Revised Report, 11.10. #
BEGIN # Formula manipulation #

  MODE FORM =
    UNION (REF CONST, REF VAR, REF TRIPLE, REF CALL),
    CONST = STRUCT (REAL value),
    VAR = STRUCT (STRING name, REAL value),
    TRIPLE = STRUCT (FORM left operand, INT operator,
      FORM right operand),
    FUNCTION = STRUCT (REF VAR bound var, FORM body),
    CALL = STRUCT (REF FUNCTION function name, FORM
      parameter);

  INT plus = 1, minus = 2, times = 3, by = 4, to = 5;
  HEAP CONST zero, one; value OF zero:= 0; value OF one:= 1;

  OP =  = (FORM a, REF CONST b) BOOL:
    CASE a IN (REF CONST ec): ec :=: b OUT FALSE ESAC;

  OP + = (FORM a, b) FORM:
    (a = zero | b |: b = zero | a | HEAP TRIPLE:= (a, plus, b));

  OP - = (FORM a, b) FORM:
     (b = zero | a | HEAP TRIPLE:= (a, minus, b));

  OP * = (FORM a, b) FORM:
  (a = zero OR b = zero | zero |: a = one | b |:
    b = one | a | HEAP TRIPLE:= (a, times, b));

  OP / = (FORM a, b) FORM:
    (a = zero AND NOT (b = zero) | zero |:
      b = one | a | HEAP TRIPLE:= (a, by, b));

  OP ** = (FORM a, REF CONST b) FORM:
    (a = one OR (b:=:zero) | one |: b:=:one | a |
      HEAP TRIPLE:= (a, to, b));

  PROC derivative of =
    (FORM e, # with respect to # REF VAR x) FORM:
    CASE e IN

      (REF CONST): zero,

      (REF VAR ev): (ev:=:x | one | zero),

      (REF TRIPLE et):
        CASE
          FORM u =  left operand OF et,
               v = right operand OF et;
          FORM udash = derivative of (u, # with respect to # x),
```

```
            vdash = derivative of (v, # with respect to # x);
        operator OF et
      IN
        udash + vdash,
        udash - vdash,
        u * vdash + udash * v,
        (udash - et * vdash) / v,
        (v | (REF CONST ec):
          v * u **
          (HEAP CONST c; value OF c:= value OF ec - 1; c)
          * udash
        )
      ESAC,

    (REF CALL ef):
      BEGIN REF FUNCTION f = function name OF ef,
                    FORM g = parameter     OF ef;
        REF VAR y = bound var OF f;
        HEAP FUNCTION fdash:= (y, derivative of (body OF f, y));
        (HEAP CALL:= (fdash, g)) * derivative of (g, x)
      END
  ESAC   # end derivative # ;

PROC value of = (FORM e) REAL:
  CASE e IN

    (REF CONST ec): value OF ec,

    (REF VAR ev): value OF ev,

    (REF TRIPLE et):
      CASE REAL u = value of (left  operand OF et),
                    v = value of (right operand OF et);
        operator OF et
      IN
        u + v,
        u - v,
        u * v,
        u / v,
        exp(v * ln(u))
      ESAC,

    (REF CALL ef):
      BEGIN REF FUNCTION f = function name OF ef;
        value OF bound var OF f:=
          value of (parameter OF ef);
        value of (body OF f)
      END
  ESAC # value of #;

HEAP FORM f, g;
```



```
  HEAP VAR a:= ("a", SKIP),
           b:= ("b", SKIP),
           x:= ("x", SKIP);
 #  start here: read ((value OF a, value OF b, value OF x));  #
  value OF a:= 1;
  value OF b:= 1;
  value OF x:= 1;

  f:= a + x/(b+x); g:= (f+ one)/(f-one);

  print((value OF a, #1#
         value OF b, #1#
         value OF x, #1#
         value of (derivative of (g, # with respect to # x))
                      #-2#))
END
```

- - . - -

```
#app107#
BEGIN # Tag list algorithm #

  MODE TAG =
    STRUCT(STRING tag, REF TAG chain, REF INFO info);

  [ 1 : 11 ] STRUCT(INT number, REF TAG chain) hashlist;
  FOR i TO UPB hashlist DO hash list[i]:= (0, NIL) OD;

  PROC find tag = (STRING tag) REF INFO:
  BEGIN REF STRUCT(INT number, REF TAG chain) handle =
      hash list[
        (INT h:= 0;
          FOR i TO UPB tag
          DO h:= (2 * h + ABS tag[i]) MOD UPB hashlist OD;
          h + 1)];
    REF REF TAG ptag:= chain OF handle;
    WHILE
      IF ptag :=: REF TAG (NIL)
      THEN REF REF TAG (ptag):=
        HEAP TAG:= (tag, NIL, HEAP INFO); FALSE
      ELIF tag OF ptag < tag THEN ptag:= chain OF ptag; TRUE
      ELIF tag OF ptag = tag THEN FALSE
      ELSE REF TAG (ptag):=
            (tag, HEAP TAG:= ptag, HEAP INFO);
            FALSE
      FI
    DO SKIP OD;
    info OF ptag
  END # find tag #;
```

```
PROC lex order = (PROC(STRING, INFO) VOID act) VOID:
( [1 : UPB hashlist] REF TAG entry:= chain OF hashlist;
  WHILE REF REF TAG entry1:= NIL;
    FOR i TO UPB entry
    DO REF REF TAG entry i = entry[i];
      IF entry i :/=: REF TAG (NIL) THEN
        IF ( entry1 :=: REF REF TAG (NIL) | TRUE |
            tag OF entry i < tag OF entry1) THEN
          entry1:= entry i
        FI
      FI
    OD;
    entry1 :/=: REF REF TAG (NIL)
  DO act( tag OF entry1, info OF entry1);
    REF REF TAG(entry1):= chain OF entry1
  OD
);

)MMENT
ROC test = VOID :
print(("Debug;", newline));
 FOR i TO UPB hashlist
 DO REF TAG rrt:= chain OF hashlist[i];
   WHILE rrt ISNT REF TAG (NIL)
   DO print((tag OF rrt, info OF rrt, newline));
     rrt:= chain OF rrt
   OD;
 print(("End hash;", newline))
 OD;
:int(("End debug;", newline))
;
)MMENT

 MODE INFO = INT;

 find tag("aap" ):= 1;
 find tag("noot"):= 4;
 find tag("mies"):= 3;
 find tag("wim" ):= 5;
 find tag("zus" ):= 6;
 find tag("jet" ):= 2;
 print( find tag("aap" ));
 print( find tag("jet" ));
 print( find tag("mies"));
 print( find tag("noot"));
 print( find tag("wim" ));
 print( find tag("zus" ));
 print(newline);
 lex order((STRING st, INFO i) VOID:
                 print((st, i, newline)))
ID
```


- - . - -

```
#app108#
rat: # Dik Winter, 141075#
BEGIN #Handling of rationals#
    MODE RAT = STRUCT (INT n, d);

    #Preliminary routines#
    OP GCD = (INT i, j) INT:
        IF i = 0
        THEN ABS j
        ELIF j = 0
        THEN ABS i
        ELSE INT ii:= ABS i, jj:= ABS j, k;
        l1: k:= ii - ii % jj * jj; ii:= jj; jj:= k;
            IF jj = 0 THEN ii ELSE l1 FI
        FI;
    PRIO GCD= 8;
    OP / = (INT i, j) RAT:
        BEGIN INT k = i GCD j;
            IF j >= 0
            THEN (i % k, j % k)
            ELSE (- i % k, - j % k)
            FI
        END;
    OP INV = (INT i) RAT:
        IF i >= 0 THEN (1, i) ELSE (- 1, - i) FI;

    #Basic operators#
    OP INV = (RAT q) RAT:
        IF n OF q >= 0
        THEN (d OF q, n OF q)
        ELSE (- d OF q, - n OF q)
        FI;
    OP + = (RAT q) RAT: q;
    OP - = (RAT q) RAT: (- n OF q, d OF q);
    OP ABS = (RAT q) RAT: (ABS n OF q, d OF q);
    OP + = (RAT q, p) RAT:
        BEGIN INT k = d OF q GCD d OF p;
            INT dq = d OF q % k, dp = d OF p % k;
            INT n = n OF q * dp + n OF p * dq;
            INT l = n GCD k, d = dp * dq;
            (n % l, k % l * d)
       END;
    OP - = (RAT q, p) RAT:
        BEGIN INT k = d OF q GCD d OF p;
            INT dq = d OF q % k, dp = d OF p % k;
            INT n = n OF q * dp - n OF p * dq;
            INT l = n GCD k, d = dp * dq;
            (n % l, k % l * d)
```

```
    END;
OP * = (RAT q, p) RAT:
    BEGIN INT nq = n OF q, np = n OF p;
        INT dq = d OF q, dp = d OF p;
        INT i = nq GCD dp, j = np GCD dq;
        ((nq % i) * (np % j), (dq % j) * (dp % i))
    END;
OP / = (RAT q, p) RAT:
    BEGIN INT nq = n OF q, np = n OF p;
        INT dq = d OF q, dp = d OF p;
        INT i = nq GCD np, j = dp GCD dq;
        IF np >= 0
        THEN ((nq % i) * (dp % j), (dq % j) * (np % i))
        ELSE (- (nq % i) * (dp % j), - (dq % j) * (np % i))
        FI
    END;
OP +:= = (REF RAT q, RAT p) REF RAT:
    BEGIN INT k = d OF q GCD d OF p;
        INT dq = d OF q % k, dp = d OF p % k;
        INT n = n OF q * dp + n OF p * dq;
        INT l = n GCD k, d = dp * dq;
        q:= (n % l, k % l * d)
   END;
OP -:= = (REF RAT q, RAT p) REF RAT:
    BEGIN INT k = d OF q GCD d OF p;
        INT dq = d OF q % k, dp = d OF p % k;
        INT n = n OF q * dp - n OF p * dq;
        INT l = n GCD k, d = dp * dq;
        q:= (n % l, k % l * d)
    END;
OP *:= = (REF RAT q, RAT p) REF RAT:
    BEGIN INT nq = n OF q, np = n OF p;
        INT dq = d OF q, dp = d OF p;
        INT i = nq GCD dp, j = np GCD dq;
        q:= ((nq % i) * (np % j), (dq % j) * (dp % i))
    END;
OP /:= = (REF RAT q, RAT p) REF RAT:
    BEGIN INT nq = n OF q, np = n OF p;
        INT dq = d OF q, dp = d OF p;
        INT i = nq GCD np, j = dp GCD dq;
        q:= IF np >= 0
            THEN ((nq % i) * (dp % j), (dq % j) * (np % i))
            ELSE (- (nq % i) * (dp % j),
                - (dq % j) * (np % i))
            FI
    END;

#Rationals mixed with integers#
OP + = (RAT q, INT i) RAT:
    (n OF q + d OF q * i, d OF q);
OP - = (RAT q, INT i) RAT:
```



```
    (n OF q - d OF q * i, d OF q);
OP * = (RAT q, INT i) RAT:
    BEGIN INT dq = d OF q; INT k = dq GCD i;
        (i % k * n OF q, dq % k)
    END;
OP / = (RAT q, INT i) RAT:
    BEGIN INT nq = n OF q; INT k = nq GCD i;
        IF i >= 0
        THEN (nq % k, i % k * d OF q)
        ELSE (- nq % k, - i % k * d OF q)
        FI
    END;
OP +:= = (REF RAT q, INT i) REF RAT:
    q:= (n OF q + d OF q * i, d OF q);
OP -:= = (REF RAT q, INT i) REF RAT:
    q:= (n OF q - d OF q * i, d OF q);
OP *:= = (REF RAT q, INT i) REF RAT:
    BEGIN INT dq = d OF q; INT k = dq GCD i;
        q:= (i % k * n OF q, dq % k)
    END;
OP /:= = (REF RAT q, INT i) REF RAT:
    BEGIN INT nq = n OF q; INT k = nq GCD i;
        q:= IF i >= 0
            THEN (nq % k, i % k * d OF q)
            ELSE (- nq % k, - i % k * d OF q)
            FI
    END;
OP + = (INT i, RAT q) RAT:
    (i * d OF q + n OF q, d OF q);
OP - = (INT i, RAT q) RAT:
    (i * d OF q - n OF q, d OF q);
OP * = (INT i, RAT q) RAT:
    BEGIN INT dq = d OF q; INT k = dq GCD i;
        (i % k * n OF q, dq % k)
    END;
OP / = (INT i, RAT q) RAT:
    BEGIN INT nq = n OF q; INT k = nq GCD i;
        IF  nq >= 0
        THEN (i % k * d OF q, nq % k)
        ELSE (- i % k * d OF q, - nq % k)
        FI
    END;

#Rationals mixed with reals#
OP VAL = (RAT q) REAL:
    REAL (n OF q) / REAL (d OF q);
OP + = (REAL r, RAT q) REAL: r + VAL q;
OP - = (REAL r, RAT q) REAL: r - VAL q;
OP * = (REAL r, RAT q) REAL: r * VAL q;
OP / = (REAL r, RAT q) REAL: r / VAL q;
OP +:= = (REF REAL r, RAT q) REF REAL: r +:= VAL q;
```

```
OP -:= = (REF REAL r, RAT q) REF REAL: r -:= VAL q;
OP *:= = (REF REAL r, RAT q) REF REAL: r *:= VAL q;
OP /:= = (REF REAL r, RAT q) REF REAL: r /:= VAL q;
OP + = (RAT q, REAL r) REAL: VAL q + r;
OP - = (RAT q, REAL r) REAL: VAL q - r;
OP * = (RAT q, REAL r) REAL: VAL q * r;
OP / = (RAT q, REAL r) REAL: VAL q / r;

#Comparing rationals#
OP = = (RAT q, p) BOOL:
    n OF q = n OF p AND d OF q = d OF p;
OP /= = (RAT q, p) BOOL:
    n OF q /= n OF p OR d OF q /= d OF p;
OP >= = (RAT q, p) BOOL:
    n OF q * d OF p >= n OF p * d OF q;
OP > = (RAT q, p) BOOL:
    n OF q * d OF p > n OF p * d OF q;
OP < = (RAT q, p) BOOL:
    n OF q * d OF p < n OF p * d OF q;
OP <= = (RAT q, p) BOOL:
    n OF q * d OF p <= n OF p * d OF q;

#Comparing rationals with integers#
OP = = (RAT q, INT i) BOOL:
    n OF q = i AND d OF q = 1;
OP /= = (RAT q, INT i) BOOL:
    n OF q /= i OR d OF q /= 1;
OP >= = (RAT q, INT i) BOOL:
    n OF q >= i * d OF q;
OP > = (RAT q, INT i) BOOL:
    n OF q > i * d OF q;
OP < = (RAT q, INT i) BOOL:
    n OF q < i * d OF q;
OP <= = (RAT q, INT i) BOOL:
    n OF q <= i * d OF q;
OP = = (INT i, RAT q) BOOL:
    i = n OF q AND d OF q = 1;
OP /= = (INT i, RAT q) BOOL:
    i /= n OF q OR d OF q /= 1;
OP >= = (INT i, RAT q) BOOL:
    i * d OF q >= n OF q;
OP > = (INT i, RAT q) BOOL:
    i * d OF q > n OF q;
OP < = (INT i, RAT q) BOOL:
    i * d OF q < n OF q;
OP <= = (INT i, RAT q) BOOL:
    i * d OF q <= n OF q;

#Comparing rationals with reals#
OP = = (REAL r, RAT q) BOOL: r = VAL q;
OP /= = (REAL r, RAT q) BOOL: r /= VAL q;
```



```
OP >= = (REAL r, RAT q) BOOL: r >= VAL q;
OP > = (REAL r, RAT q) BOOL: r > VAL q;
OP < = (REAL r, RAT q) BOOL: r < VAL q;
OP <= = (REAL r, RAT q) BOOL: r <= VAL q;
OP = = (RAT q, REAL r) BOOL: VAL q = r;
OP /= = (RAT q, REAL r) BOOL: VAL q /= r;
OP >= = (RAT q, REAL r) BOOL: VAL q >= r;
OP > = (RAT q, REAL r) BOOL: VAL q > r;
OP < = (RAT q, REAL r) BOOL: VAL q < r;
OP <= = (RAT q, REAL r) BOOL: VAL q <= r;

#Converting rationals to a number string#
PROC rat = (RAT q, INT width) STRING:
    IF STRING s = (q < 0 | "-(" |: width > 0 | "+(" | "(") +
        whole(ABS n OF q, 0) + "/" + whole(d OF q, 0) + ")";
        width = 0
    THEN s
    ELSE IF INT us = UPB s, aw = ABS width;
            us > aw
        THEN aw * (q < 0 | "-" | "+")
        ELSE (aw - us) * " " + s
        FI
    FI;

#Innerproduct of two arrays of rationals#
OP +* = (REF [] RAT a, b) RAT:
    BEGIN RAT s:= (0, 1);
        FOR i TO UPB a
        DO s+:= a[i] * b[i]
        OD;
        s
    END;

#LU-decomposition of a matrix of rationals#
PROC decrat = (REF [,] RAT a, REF [] INT p) VOID:
    BEGIN INT n = 1 UPB a;
        FOR k TO n
        DO RAT piv:= (0,1), INT k1:= k - 1;
            REF INT pk = p[k];
            REF [] RAT aik = a[,k], aki = a[k,];
            FOR i FROM k TO n
            DO aik[i]-:= a[i,1:k1] +* aik[1:k1];
                IF piv = 0 AND aik[i] /= 0
                THEN piv:= aik[i]; pk:= i
                FI
            OD;
            IF piv = 0
            THEN print((newline, newline, "Singular matrix"));
                stop
            FI;
            IF pk /= k
```

```
            THEN FOR i TO n
                DO RAT r = aki[i];
                    aki[i]:= a[pk,i]; a[pk,i]:= - r
                OD
            FI;
            FOR i FROM k + 1 TO n
            DO aki[i]-:= aki[1:k1] +* a[1:k1,i] /:= piv
            OD
        OD
    END;

  #Calculation of the determinant of a decomposed matrix#
  PROC determrat = (REF [,] RAT a) RAT:
      BEGIN RAT d:= (1, 1);
          FOR i TO 1 UPB a
          DO d*:= a[i,i]
          OD;
          d
      END;

  FOR n TO 5
  DO [1:n,1:n] RAT a;
      FOR i TO n
      DO a[i,i]:= INV (i * 2 - 1);
          FOR j FROM i + 1 TO n
          DO a[i,j]:= a[j,i]:= INV (i + j - 1)
          OD
      OD;
      decrat(a, LOC [1:n] INT);
      print(("Order: ", whole(n, - 1), "; determinant: ",
      rat(determrat(a), 0), newline))
  OD
ND
```

- - . - -

```
ippl09#
EGIN  # Bubble sort, ALGOL 68 version #

PROC sort= (REF [] INT a) VOID:
BEGIN INT p = LWB a;
  FOR dp FROM p+1 TO UPB a DO
    INT bp:= dp; INT bubble= a[bp];
    WHILE INT prev;
      IF bp = p THEN FALSE ELSE
        prev:= a[bp-1]; prev > bubble FI
    DO    a[bp]:= prev; bp-:= 1 OD;
    a[bp]:= bubble
  OD
END # sort #;
```



```
  PROC  shuffle= (REF [] INT a) VOID:
  BEGIN INT p= LWB a, q= UPB a;
    FOR i FROM q BY -1 TO p+1
    DO REF INT t= a[ENTIER (random * (i-p+1)) + p], u = a[i];
      INT h= t; t:= u; u:= h # swap #
    OD
  END # shuffle #;

  INT max=  8; [ 1 : max ] INT p;

  PROC  test= (PROC (INT) INT a) VOID:
  ( FOR i TO max DO p[i]:= a(i)OD;
    shuffle(p); print(("Shuffled:",newline,p,newline));
    sort(p); print(("Sorted:",newline,p,newline,newline))
  );

  test((INT p) INT: p);
  test((INT p) INT: ENTIER (p/5));
  test((INT p) INT: 0)
END
```

- - . - -

```
#appl10#
BEGIN # Conversion from Gregorian date to weekday #

    PROC weekday = (INT year, month, day) STRING:
    []STRING("Sun", "Mon", "Tues", "Wednes", "Thurs", "Fri", "Satur")
        [( INT y:= year, m:= month - 2;
            IF m < 1 THEN m +:= 12; y -:= 1 FI;
                # since the year actually starts March 1st #
            365 * y                  # number of days since year 0 #
            + y OVER 4               # plus leap days #
            - y OVER 100             # minus 1 for every century #
            + y OVER 400             # plus 1 for every 4 centuries #
            + []INT
             (0, 31, 61, 92, 122, 153, 184, 214, 245, 275, 306, 337) [m]
                    # plus number of days in this year since March 1 #
            + day + 2) MOD 7 + 1
        ] + "day";

    # Prints the week around 1968, Feb 29th, starting at Sunday #
    print((weekday(1968, 2, 25), newline));
    print((weekday(1968, 2, 26), newline));
    print((weekday(1968, 2, 27), newline));
    print((weekday(1968, 2, 28), newline));
    print((weekday(1968, 2, 29), newline));
    print((weekday(1968, 3,  1), newline));
    print((weekday(1968, 3,  2), newline))
```

```
ND
```

- - . - -

```
appl11#
OMMENT This is the intended program
EGIN # ALGOL 68 program TJD730702.
       This program tests some operator calculus and print the same
       results as the ALGOL 68 program TJD730701, viz.,
       a difference table of a 4-th degree polynomial. #

 MODE FUN = PROC(INT)INT;

 OPERATOR nabla = (FUN t)FUN :
 (INT x)INT : t (x)- t (x-1);

 MODE OPERATOR = PROC(FUN)FUN;

 OP UP = (OPERATOR a, INT b)OPERATOR :
 (FUN f) FUN :
     IF b=0 THEN f
     ELSE a ( (a UP (b-1)) (f)) FI;


 PRIO MIN = 1;
 OP MIN = (INT a, b)INT : (a<=b | a | b);
 FUN pol4 = (INT x)INT : x*(x+1)*(x+2)*(x+3);


 FOR n FROM 0 TO 20
 DO
   print(n);
   FOR k FROM 0 TO (n-1) MIN 5
   DO
     print((nabla UP k) (pol4) (n))
   OD;
   print(newline)
 OD

ND
OMMENT # end of intended program #

EGIN # Attempt at partial parametrization #

 # A / between ## separates the partial params from the direct ones #

 MODE FUN = UNION(PROC(INT)INT, FUNINTINT);
 MODE FUNINTINT = # caused by nabla #
   STRUCT(PROC(FUN, # / # INT)INT f, REF FUN p);
```



```
  OP FUN2INT = (FUN f, INT i)INT :
  CASE f IN
    (PROC(INT)INT pf) : pf(i),
    (FUNINTINT f) : (f OF f)(p OF f, i)
  ESAC;

  OPERATOR nabla = (FUN t)FUN :
  FUNINTINT # cast for scope-violating object #
  ( (FUN t, INT x)INT : t FUN2INT (x)- t FUN2INT (x-1),
       HEAP FUN:= t);

  MODE OPERATOR = UNION(PROC(FUN)FUN, OPINTFUN);
  MODE OPINTFUN = # caused by UP #
    STRUCT( PROC(OPERATOR, INT, # / # FUN) FUN f,
      REF OPERATOR p1, REF INT p2);

  OP OP2FUN = (OPERATOR op, FUN f) FUN :
  CASE op IN
    (PROC(FUN)FUN pf) : pf(f),
    (OPINTFUN op) : (f OF op)(p1 OF op, p2 OF op, f)
  ESAC;

  OP UP = (OPERATOR a, INT b)OPERATOR :
  OPINTFUN # cast for scope-violating object #
  ( (OPERATOR a, INT b, # / # FUN f) FUN :
      IF b=0 THEN f
      ELSE a OP2FUN ( (a UP (b-1)) OP2FUN (f)) FI,
    HEAP OPERATOR:= a, HEAP INT:= b);


  PRIO MIN = 1;
  PRIO FUN2INT = 9;
  PRIO OP2FUN = 9;
  OP MIN = (INT a, b)INT : (a<=b | a | b);
  FUN pol4 = (INT x)INT : x*(x+1)*(x+2)*(x+3);


  FOR n FROM 0 TO 20
  DO
    print(n);
    FOR k FROM 0 TO (n-1) MIN 5
    DO
      print((nabla UP k) OP2FUN (pol4) FUN2INT (n))
    OD;
    print(newline)
  OD

END
```

- - . - -

```
#appl12#
BEGIN # AvW, 1974:10:23, packing small integers into a larger integer#

      PROC code = ([]INT sequence) INT:
        (INT code:= 0;
         FOR k TO UPB sequence
         DO code *:= 2 +:= 1 *:= 2 ** sequence[k] OD;
         code),

      PROC length = (INT code) INT:
        (INT length:= 0, c:= code;
         WHILE c > 0
         DO (ODD c | length +:= 1); c OVERAB 2 OD;
         length),

      PROC sequence = (INT code) []INT:
        (INT l:= length(code), c:= code;
         [ 1 : l ] INT sequence;
         FOR k TO l DO sequence[k]:= 0 OD;
         WHILE c > 0
         DO (ODD c | l-:= 1 | sequence[l]+:= 1); c OVERAB 2 OD;
         sequence);

      FOR c FROM 0 TO 100
      DO print((c, length(c), sequence(c), newline,
            code(sequence(c)), newline, newline))
      OD
END
```

- - . - -

```
#appl13#
          #   Ring code   #
BEGIN INT m:=4; m:=2**m; INT n # left-most bit # = m OVER 2;
     [0:m-1] BOOL f; [1:m] INT t;
     FOR i FROM 0 TO m-1 DO f[i]:=TRUE OD;
     PROC p=(INT i,k) VOID:
       BEGIN t[k]:=i; f[i]:=FALSE;
          IF k=m
          THEN print(newline);
                # Every bit column in 't' now contains the ring code #
                FOR k TO m DO print(t[k] GE n)OD
          ELSE INT l;
             IF f[l:=2*i MOD m] THEN p(l,k+1) FI;
             IF f[l+:=1] THEN p(l,k+1) FI
          FI;
          f[i]:=TRUE
        END;
     p(0,1)
END
```


- - . - -

```
#appl14#
     # JKok, 770822, 'n' queens on chess board
       using prepared bit patterns for forbidden fields #

FOR n FROM 3 TO 8
DO INT maxbord = n * n, nflds = (n * n - 1) OVER bits width;

   # Bit pattern prelude #
   MODE BORD = [0 : nflds] BITS;
   # BORD = [0 : maxbord - 1] BOOL packed in [ ] BITS #

   OP OR = (BORD a, b) BORD :
   ( BORD c; FOR i FROM 0 TO nflds
      DO c[i]:= a[i] OR b[i] OD; c );

   OP ELEM = (INT i, BORD a) BOOL :
   IF i < 0 OR i >= max bord THEN FALSE
   ELSE (i MOD bits width + 1) ELEM a[i OVER bits width]
   FI;

   OP BTOB = (INT i) BORD : # true -->> i-th bool of bord #
   BEGIN BORD a;
      FOR k FROM 0 TO nflds DO a[k]:= 2r0 OD;
      IF i >= 0 AND i < max bord
      THEN a[i OVER bits width]:=
         2r1 UP ((- i - 1) MOD bits width)
      FI;
      a
   END # of op bool to bits #;

   OP ORAB = (REF BORD a, INT i) REF BORD :
   BEGIN IF i >= 0 AND i < max bord
      THEN REF BITS ai = a[i OVER bits width];
         ai:= (2r1 UP ((- i - 1) MOD bits width)) OR ai
      FI;
      a
   END # of op or a b #;

   PRIO ORAB = 1;


   # Initialize #
   INT aantal := 0, [1 : n] INT shift, [1 : n, 1 : n] BORD erase;

   FOR i TO n DO shift[i]:= (i - 1) * n - 1 OD;

   FOR r TO n
   DO FOR k TO n
```

```
   DO BORD ds:= BTOB -1;
      FOR i TO n - r
      DO INT sh = shift[i + r] + k; ds ORAB sh;
         IF k + i <= n THEN ds ORAB sh + i FI;
         IF k > i THEN ds ORAB sh - i FI
      OD;
      erase[r, k]:= ds
   OD
OD;


# Find all solutions #
[1 : n] INT dame;

PROC zet = (INT row, col, BORD stand) VOID :
IF dame[row]:= col; row = n THEN out sol
ELSE INT r = row + 1, sh = shift[r], damel = dame[1];
   FOR k FROM ( damel = 1 | 2 |: damel < r AND
         n - damel > r - 2 | 1 | 2 )
      TO ( damel <= r AND n - damel >= r | n | n - 1 )
   DO IF NOT ((k + sh) ELEM stand)
      THEN zet(r, k, erase[r, k] OR stand)
      FI
   OD
FI;


PROC outsol = VOID :
BEGIN
   print((newline, " ")); PROC line = VOID :
   FOR i TO 4 * n - 1 DO print("-") OD; line;
   FOR i TO n
   DO INT k = dame[i]; print(newline);
      FOR j TO n
      DO print(IF j = k THEN "| q " ELSE "|   " FI)
      OD;
      print(("|", newline, " ")); line
   OD;
   print((newline, " #", whole(aantal +:= 1, -4)));
   TO 2 DO print(newline) OD
END # out solution # ;


FOR i TO n OVER 2 DO zet(1, i, erase[1, i]) OD;

print((" Number of solutions is ", whole(aantal, -4),
   "  for n = ", whole(n, -3), newline, newline))
D
```

- - . - -


```
#appl15#
BEGIN # Mincer #
CO
This program operates in one of two modes:
     1.  garbage in, garbage out
     2.  data in, garbage out
The basic idea is to read in a program, scramble it, and punch the
scrambled program out. The scrambled program can be fed into a compiler
to see what it does. Experience shows that most compilers do not take
well to this test at all.

The program is broken up into syntactic units, where a syntactic unit
is an identifier, an unsigned int, a bold, a string, a sequence of
the characters +-*/=<>: , or a special.

Random numbers are taken from a rectangular distribution with mean
supplied by the user. Let 'n1' be the first such random number.
The first 'n1' syntactic units are considered to be the first chunk. The
next 'n2' syntactic units comprise the second chunk, etc. The chunks are
then put out in random order. If the chunks are big enough, the compiler
thinks it is getting reasonable stuff, makes some attempt at analyzing
the structure, building tables, etc. If the chunks are too small,
nothing much happens.

The program to be read in resides on the file "program", the scrambled
program on the file "result".
The values to be used as means for the random numbers are read from
'stand in'. The list is terminated by mean = 0. Values may be
preceded by a minus-sign, in which case the chunks in the corresponding
output are separated by newlines.
For instance, if the input-file contains:  100   -20   0  ,
two scrambled programs will be generated, the first having chunks of
about 100 syntactic units, the second with chunks of about 20 syntactic
units, separated.
CO

  INT line width = 72; # for 'program' and 'result' #
  FILE program; # contains the program #
  open(program, "program", standin channel);
  FILE result; # will contain the minced program #
  establish(result, "result", standout channel, 1, 10000, line width);

  STRING result sep = "######"; # for result #
  CHAR quote = """", dot = ".";

  PROC in item = STRING:
  (STRING st = in item or comment;
    comment(st) | skip comment(st); in item | st);

  PROC comment = (STRING s) BOOL:
  s = "#" OR s = "CO" OR s = "COMMENT";
```

```
PROC skip comment = (STRING s) VOID:
WHILE in item or comment /= s DO SKIP OD;

PROC in item or comment = STRING:
BEGIN more real input; CHAR ch = line[c pos];

  STRUCT(STRING item, INT new pos) res :=
  IF letter(ch)
  THEN INT p = last(letgit);
    (line[c pos: p], p + 1)
  ELIF ch = quote
  THEN INT p = last((CHAR c) BOOL: c /= quote);
    (line[c pos: p] + quote, p + 2)
  ELIF digit(ch)
  THEN INT p = last(digit);
    (line[c pos: p], p + 1)
  ELIF ch = dot
  THEN INT p = last(letgit);
    (line[c pos: p] + " ", p + 1)
  ELIF indicant (ch)
  THEN INT p = last(indicant);
    (line[c pos: p], p + 1)
  ELSE (line[c pos], c pos + 1)
  FI;
  c pos:= new pos OF res; item OF res
END # in item or comment #;

PROC last= (PROC (CHAR) BOOL cond) INT:
(INT p:= c pos;
  FOR d FROM c pos + 1 TO UPB line WHILE cond(line[d])
  DO p:= d OD;
  p
);

PROC letter = (CHAR ch) BOOL: "a" <= ch AND ch <= "z"
    OR "A" <= ch AND ch <= "Z" # for UPPER-style #;

PROC digit = (CHAR ch) BOOL: "0" <= ch AND ch <= "9";

PROC letgit = (CHAR ch) BOOL: letter (ch) OR digit (ch);

PROC indicant = (CHAR ch) BOOL:
  char in string (ch, LOC INT, "+-*/=<>:");

PROC more real input = VOID:
(skip:
  IF c pos>UPB line THEN newline(program); get line; skip FI;
  IF line [c pos] = " " THEN c pos +:= 1; skip FI
);

INT c pos:= 1, STRING line:= ""; # on 'program' #
```



```
  PROC get line = VOID:
  (get(program, line);
    IF UPB line > line width
    THEN line:= line [1: linewidth] FI;
    c pos:= 1
  );

  PROC out item = (STRING s) VOID:
  (IF char number (result) + UPB s > line width
    THEN newline (result) FI;
    put(result, s)
  );

  PROC range = (INT r)INT: # a random integer in the range [1:r] #
  ENTIER (random * r) + 1;

# Reading the program text #
  MODE TEXT = STRUCT (STRING string, REF TEXT next);
  REF TEXT no text = NIL;
  REF TEXT first text:= no text, last text:= no text;

  on logical file end (program, (REF FILE f) BOOL: run);

# Initialize # get line;
  DO # until end-of-file # STRING st = in item;
    last text:=
    (last text :=: no text| first text | next OF last text):=
    HEAP TEXT:= (st, no text)
  OD;

  run:
  WHILE INT descr = (INT i; read(i); i);
    INT mean = ABS descr, BOOL sep = descr < 0;
    0 < mean AND mean < 10000
  DO

    MODE CHUNK =
    STRUCT(STRUCT(INT length, REF TEXT text) chunk,
              REF CHUNK next);

    REF CHUNK no chunk = NIL;
    REF CHUNK first chunk:= no chunk, last chunk:= no chunk;
    INT n chunks:= 0; last text:= first text;

    WHILE last text :/=: no text
    DO INT cnt:= 0, REF TEXT p:= last text;

      TO range (2 * mean - 1)
      DO (p :/=: no text
           | p:= next OF p; cnt +:=1)
      OD # determine chunk #;
```

```
    # enter into chunk chain #
      last chunk:=
      (last chunk :=: no chunk
      | first chunk
      | next OF last chunk):=
      HEAP CHUNK:= ((cnt, last text), NIL);
      n chunks +:= 1; last text:= p
    OD # chunk chain ready #;

  # Tie full-circle #
    (last chunk :/=: no chunk | next OF last chunk:= first chunk);

  # Mix the chunks #
    FOR length FROM n chunks BY -1 TO 1
    DO

      TO range (length)
      DO first chunk:= next OF first chunk OD;

    # Random chunk found, now write it #
      REF TEXT p:= text OF chunk OF next OF first chunk;
      TO length OF chunk OF next OF first chunk
      DO out item (string OF p);
        p:= next OF p
      OD;

      IF sep THEN newline(result) FI;

    # Remove chunk #
      next OF first chunk:=
        next OF next OF first chunk
    OD;
    put(result, (newline, result sep, newline, newline));

    printf((
      $ "Produced" 4zd x, "chunks of mean length" 3zd ,
        b (", separated", "") 1 $,
      n chunks, mean, sep))

  OD
END
```

- - . - -

```
#appl16#
BEGIN # Sheep in mountain cleft #
  # See A. van Wijngaarden, Programmacorrectheid en grammatica's,
    in Mathematical Centre Syllabus 21, XII, 1975. #

  PROC p = (INT i, j, STRING s) VOID:
```



```
    # i is line number, j is the position of the dot in s;
      three spaces have been appended to the left and the right
      of s, in order to simplify the testing #
    BEGIN print((newline, i, " ", s[ 4 : n ]));
      IF   s[j-2:j  ] = "><."                                  # h6 #
      THEN p(i+1, j-2, s[:j-3] + ".<>" + s[j+1:])
      ELIF s[j  :j+2] = ".><"                                  # h7 #
      THEN p(i+1, j+2, s[:j-1] + "<>." + s[j+3:])
      ELIF s[j-1:j+3] = ">.<><"                                # h4 #
      THEN p(i+1, j-1, s[:j-2] + ".>" + s[j+1:])
      ELIF s[j-3:j+1] = "><>.<"                                # h5 #
      THEN p(i+1, j+1, s[:j-1] + "<." + s[j+2:])
      ELIF s[j-1:j+1] = ">.<"                                  # h4, h5
      THEN print(newline);
        p(i+1, j-1, s[:j-2] + ".>" + s[j+1:]);
        print(newline);
        p(i+1, j+1, s[:j-1] + "<." + s[j+2:])
      ELIF s[j-1:j  ] = ">."                                   # h8 #
      THEN p(i+1, j-1, s[:j-2] + ".>" + s[j+1:])
      ELIF s[j  :j+1] = ".<"
      THEN p(i+1, j+1, s[:j-1] + "<." + s[j+2:])               # h9 #
      FI
    END # p # ;

  INT a, b;
# read((a, b)); # a:= 3; b:= 3;
  INT n = a + b + 7;
  IF a >= 0 AND b >= 0
  THEN p(1, a+4, "   " + a*">" + "." + b*"<" + "   ")
  FI
END
```

- - . - -

```
#appl17#
BEGIN # All-parser, Dick Grune, 20-11-74.
        The following is an example of a technique that will give a
        parser for any non-left-recursive context-free grammar.
        The parser gives all possible parsings.
      #

  MODE ACT = VOID, TAIL = PROC ACT,
    RULE = PROC (TAIL) ACT;

  #         R u l e                          G r a m m a r #

  RULE t= (TAIL q) ACT: s(ACT: b(q));         # t: s, b.     #

  RULE s = (TAIL q) ACT:                      # s:           #
    ( a(ACT: s(ACT: s(q)));                   #    a, s, s;  #
```

```
      a(q)                                  #    a.        # ,
    );

  RULE a = (TAIL q) ACT :
    (n +:= 1; IF inp[n] = "a" THEN q FI ;      # a: "a".      #
     n -:= 1);

  RULE b = (TAIL q) ACT :
    (n +:= 1; IF inp[n] = "b" THEN q FI ;      # b: "b".      #
     n -:= 1);

  STRING inp, INT n:= 0;

  INT max = 10;
  FOR i FROM 0 TO max
  DO inp:= i * "a" + "b"; INT count:= 0;
    t(ACT: count+:=1);
    print(("The sentence """, inp, """ has", count, " parsings",
           newline))
  OD
END
```

\- - . - -

```
#appl18#
# Happy family, taken from: C.H. Lindsey and S.G. van der Meulen,
                "Informal Introduction to ALGOL 68", Revised edition #

BEGIN

COMMENT This  example  concerns  people: COMMENT

    MODE PERSON = STRUCT
            (STRING surname, given # name #,
            REF PERSON father, mother, wife # or husband #,
            FLEX [1:0] REF PERSON children,
            BOOL dead, male);
    BOOL male = TRUE, female = FALSE, alive = FALSE,
        dead = TRUE;
    REF PERSON nobody = NIL;

COMMENT Sometimes it will be convenient to have a PERSON's given name
and surname together: COMMENT

    PROC names =
        (REF PERSON pers) STRUCT(STRING given, surname):
        (given OF pers, surname OF pers);

COMMENT All our formal-parameters will be of mode REF PERSON
rather than PERSON, to save making unnecessary copies of PERSON's
```



```
(which are rather large) at run time. COMMENT

COMMENT Here is a procedure that will be used to add a little random
spice to the messages that we shall produce. It yields a random integ
in the range specified by its parameter. COMMENT

    PROC randint = (INT range) INT:
        1 + ENTIER (random*range);
    read(last random); # to start it off #

COMMENT This program is going to print texts of variable length. We
therefore have to take a newline whenever a line is full (after 80
characters, say), but before doing this we must go back to the last
space and transfer the whole of the word which was about to be split
onto the next line.  Therefore, we shall output into a []CHAR instead
of directly to the book. COMMENT

    FILE file; [1:80] CHAR buffer;
    FOR i TO UPB buffer DO buffer[i]:= " " OD;
    associate(file, buffer);

COMMENT Whenever the buffer becomes full, its contents (except for th
split word) must be printed in the real book. COMMENT

    PROC empty buffer = (REF FILE f) BOOL:
        (INT j:= UPB buffer;
        IF char number(f) > j
        THEN WHILE buffer[j] /= " " DO j -:= 1 OD
        FI;
        print((buffer[ :j], newline));
        reset(f);
        put(f, buffer[j+1: ]);
        FOR i FROM UPB buffer -j+1 TO UPB buffer
        DO buffer[i]:= " " OD;
        TRUE);
    on line end(file, empty buffer);

COMMENT The []CHAR associated with 'file' is like a book containing
one page containing one line. As soon as we call 'newline(file)',
therefore, we shall find that the page has overflowed (the current
position will actually be at '(1,2,1)'). COMMENT

    on page end(file, empty buffer);
    STRUCT (INT day, [1:3] CHAR month, INT year) date;

COMMENT We shall frequently have occasion to print dates. Here is a
FORMAT to do it. COMMENT

    FORMAT datef = $ g(0)x, 3ax, 2d $;
    PROC generate =
        (REF PERSON infant, father, mother,
```

```
        STRING given name, BOOL male) VOID:
  IF male OF father
      AND NOT male OF mother AND NOT dead OF mother
  THEN
      OP PLUSAB =
      (REF FLEX[]REF PERSON names, REF PERSON pers) VOID:
      names:= (INT upb = UPB names;
          [1:upb + 1]REF PERSON new names;
          new names[1:upb] :=names;
          new names[upb + 1]:= pers; new names);
      infant:= (surname OF mother,
          given name,
          father,
          mother,
          NIL,
          (), # not yet! #
          alive,
          male);
      children OF father PLUSAB infant;
      children OF mother PLUSAB infant;
      IF wife OF father :=: mother

)MMENT That was an identity relation. If you have not yet read 5.7.4,
lease accept our assurance that ':=:' is a sort of operator which
ields TRUE if the two names which are its operands in fact are the
ame name.  In this case, the operands were of mode REF PERSON, and
? the PERSON's REF'ed to turn out to be the same PERSON COMMENT

      THEN
          putf(file,
            ( $21"Birth."
                 1 4x g$,          surname OF infant,
              $". On " f(datef)$, date,
              $" to "g$,          given OF mother,
              $", wife of "g$,    given OF father,
              $", a "c("darling",
                       "bouncing",
                       "beautiful",
                       "tiny")$,  randint(4),
              $x b("son", "daughter")
              " - "$,             male,
              $g"."$,             given name))

)MMENT ELSE no comment COMMENT

      FI

)MMENT The above call of 'putf' is intended to produce messages such
:
   Birth.
       Fitzwilliam. On 3 MAR 28 to Eleanor, wife of
```



```
       Ebenezer, a beautiful son - Japhet. COMMENT

       ELSE stop # the birth was quite impossible #
       FI; # end of generate #

COMMENT The following procedure is intended to print the name of some
PERSON, together with details of his parents. However, if there is
some doubt about the marital state of the parents, then we shall draw a
discreet veil over the matter by using a different FORMAT. COMMENT

       PROC details = (REF PERSON pers) VOID:
           IF mother OF pers :=:
                   REF PERSON (wife OF father OF pers)
           THEN
               BOOL sex = male OF pers;
               putf(file,
                   ($ g ", " $,           given OF pers,
                       $ c("only", "youngest", "younger",
                           "eldest", "elder", "") x $,
                       (INT j:= 0, k;
                       REF FLEX [] REF PERSON children =
                               children OF father OF pers;
                       INT upb = UPB children;
                       FOR i TO upb # each brother/sister of pers #
                       DO REF PERSON child = children[i];
                               (male OF child = sex | j+:= 1);
                               (given OF child = given OF pers | k:= j)
                       OD;
                       (j=1 | 1 # only #
                       |: k=j | 2+ ABS (j=2) # youngest or younger #
                       |: k=1 | 4+ ABS (j=2) # eldest or elder #
                       | 6)),
                   $ b("son", "daughter")
                       " of " $,          sex,
                   $ g " and ", g x, g $, given OF father OF pers,
                                          names(mother OF pers) ))
               ELSE putf(file, ($ g x, g $, names(pers)))
               FI;  # end of details #
           PROC marry = ( REF PERSON bride, groom) VOID:
               IF
                   male OF groom AND NOT dead OF groom
                   AND NOT male OF bride AND NOT dead OF bride
                   AND   (wife OF groom :=: nobody | TRUE
                         | dead OF wife OF groom)
                   AND   (wife OF bride # sic # :=: nobody | TRUE
                         | dead OF wife OF bride)
               THEN
                   wife OF groom:= bride;
                   wife OF bride:= groom;

COMMENT We are now going to produce a message such as:
```

```
Marriage.
    Fitzwilliam/Jones. On 1 APR 24, Eleanor, only
daughter of Emrys and Myfanwy Jones to Ebenezer,
elder son of Aloysius and Anastasia Fitzwilliam.  COMMENT

            putf(file,
                ($ 21 "Marriage."
                    l 4x g "/", g". On " $,
                    surname OF groom, surname OF bride,
                $ f(datef) ", " $, date));
            details(bride);
            put(file, " to ");
            details(groom);
            put(file, ".");
            surname OF bride:= surname OF groom
        ELSE stop
                # the marriage is impossible, or illegal, or both #
        FI;  # end of marry #
    PROC kill = (REF PERSON bloke) VOID:
        IF NOT dead OF bloke
        THEN
            dead OF bloke:= TRUE;
            BOOL sex = male OF bloke;
            BOOL wa # wife alive # =
                (wife OF bloke :=: nobody | FALSE
                | NOT dead OF wife OF bloke);
            STRING # name of # wife =
                            (wa | given OF wife OF bloke | "" );

)MMENT The following call of 'putf' is intended to produce messages
ıch as:

Death.
    On 21 DEC 68, Ebenezer, elder son of Aloysius
and Anastasia Fitzwilliam, mourned by his devoted
wife Eleanor COMMENT

            putf(file,
            ( $ 21 "Death."
                14x "On " f(datef) ", " $, date));
            details(bloke);
            IF wa
            THEN
                putf(file,
                    ( $", mourned by "
                    b("his", "her") x,
                    c("everloving", "devoted",
                        "thankful") x,
                    b("wife", "husband"),
                    x g $,
                    sex, randint(3), sex, wife))
```



```
                FI;

COMMENT If 'bloke' has surviving descendants, the dirge continues in
the following vein:

    and his children Shem, Ham and Japhet and his
  grandchildren Ananias, Azarias and Misael and his
  great-grandchild Tom. COMMENT

                BOOL mp # mourners printed # := wa;

COMMENT The following PROC calls itself recursively for each
generation. COMMENT

                PROC print children of = ([]REF PERSON parents,
                        INT generation) VOID:
                BEGIN INT i:=0, j:=0;
                    [1: (INT i := 0;
                        FOR j TO UPB parents
                        DO i +:= UPB children OF parents[j] OD;
                        i) ] REF PERSON children, living children;
                    FOR k TO UPB parents
                    DO FOR l TO UPB children OF parents[k]
                        DO REF PERSON child =
                                (children OF parents[k]) [l];
                            children[i +:= 1] :=
                                (NOT dead OF child
                                |living children[j +:= 1] := child
                                |child)
                        OD
                    OD;
                    IF j /= 0
                    THEN # there are living children to be printed #
                        putf(file,
                        ( $ f(mp | $ " and" $ |: wa | $ "," $
                                 | $ "mourned by" $),
                            x b("his", "her") x,
                            n(generation-1) "great-"
                            f(generation /= 0 | $ "grand" $ | $ $),
                            "child" f(j /= 1 | $ "ren" $ | $ $) x,
                            n(j) (g, f((j-:=1) + 1
                                        | $ $, $ " and " $
                                        | $ ", " $))$,
                          sex,
                          ( [1:j] STRING names;
                            FOR i TO j
                            DO names[i]:=
                                    given OF living children[i]
                            OD;
                            names) ));
                    mp:= TRUE
```

```
            FI;
            IF UPB children /= 0
            THEN print children of(children, generation + 1)
            FI
        END # of print children of #;
        print children of(bloke, 0);
        put(file, ".")
    ELSE stop # the bloke was dead already #
    FI # end of kill #;

)MMENT Now we are ready to start our tale. Since we do not wish to go
ight back to Adam, we shall start by declaring the story so far:
)MMENT

  PERSON aloysius :=
      ("Fitzwilliam", "Aloysius", SKIP, SKIP, SKIP, (),
          dead, male);
  PERSON anastasia :=
      ("Fitzwilliam", "Anastasia", SKIP, SKIP, aloysius, (),
          dead, female);
  PERSON ebenezer :=
      ("Fitzwilliam", "Ebenezer", aloysius, anastasia, NIL, (),
          alive, male);
  PERSON alaric :=
      ("Fitzwilliam", "Alaric", aloysius, anastasia, NIL, (),
          alive, male);

)MMENT We were unable to include 'anastasia' as 'aloysius'' wife when
itialising him, because her declaration had not been elaborated at
iat time (cf. 3.2.E7).  We can rectify this, and the similar case of
eir children, now COMMENT

  wife OF aloysius := anastasia;
  children OF aloysius := children OF anastasia :=
      (ebenezer, alaric);

)MMENT We shall declare the next family differently, so avoiding this
:oblem: COMMENT

  PERSON emrys, myfanwy, frederick, eleanor;
  emrys:= ("Jones", "Emrys", SKIP, SKIP, myfanwy,
                  (frederick, eleanor), dead, male);
  myfanwy:= ("Jones", "Myfanwy", SKIP, SKIP, emrys,
                  children OF emrys, alive, female);
  frederick:= ("Jones", "Frederick", emrys, myfanwy, NIL, (),
                  alive, male);
  eleanor:= ("Jones", "Eleanor", emrys, myfanwy, NIL, (),
                  alive, female);
  PERSON shem, ham, japhet, ananias, azarias, misael, tom;

)MMENT These are the unborn generations, and are therefore undefined.
```



```
COMMENT

    date := (1, "APR", 24); marry(eleanor, ebenezer);
    date := (1, "JAN", 25); generate(shem, ebenezer, eleanor,
                "Shem", male);

COMMENT We don't waste much time in this program. COMMENT

    date := (31, "MAR", 26); generate(ham, ebenezer, eleanor,
                "Ham", male);
    date := (3, "MAR", 28); generate(japhet, ebenezer, eleanor,
                "Japhet", male);

COMMENT This will produce the example given in the PROC 'generate'.
COMMENT

    date := (14, "JUL", 48);

COMMENT Now we need to declare some eligible young ladies. COMMENT

    PERSON a, b, josie, rosie;
    josie := ("Smith", "Josephine", a, b, NIL, (), alive, female);
    rosie := ("Smith", "Rose", a, b, NIL, (), alive, female);
    marry(josie, shem);
    date := (23, "JAN", 49); generate(ananias, shem, josie,
                "Ananias", male);

COMMENT Well, perhaps it was premature. COMMENT

    date := (14, "DEC", 50); generate(azarias, shem, josie,
                "Azarias", male);
    date := (29, "FEB", 52); kill(josie);

COMMENT Alas! But ... COMMENT

    date := (28, "DEC", 52); marry(rosie, shem);

COMMENT There are some interesting ecclesiastical problems in that
one. COMMENT

    date := (14, "JAN", 54); generate(misael, shem, rosie,
                "Misael", male);

COMMENT Here is a not-so-eligible young lady: COMMENT

    PERSON x :=
        (CHAR(SKIP), CHAR(SKIP), SKIP, SKIP, NIL,
            REF PERSON(SKIP), alive, female);
    date := (20, "DEC", 68); generate(tom, azarias, x, "Tom", male);

COMMENT And so the permissive society has arrived. Nothing will be
```

```
printed. COMMENT

    date := (21, "DEC", 68); kill(ebenezer);

COMMENT Poor chap! this will produce the example given in the PROC
'kill'. COMMENT

    newline(file); newline(file) # to ensure that the final contents
                                   of the buffer get printed #
END
```

- - . - -

```
#appl19#
#
  +prpl>lftc   "input"   "tables"
"forward"
a: "x", "d";
   b, "c";
"e", "x", "c";
   "e", b, "d";
   "u", c.
b: "x".
c: c.
d: c.
a: a.
.
#
# The above is input for this program #
BEGIN
   # For timing information see routine 'time', near line 100 #
STRING progname = "Parser Generator for ALGOL 68 H -- Version 2.0.0";

   # This is an intermediate version of a program being
     written by Hendrik Boom. It should not be considered
     to be a finished product; however, this present
     version appears to work.
   #
   # <><><><>
      Things to do:
      Find out whether it is worth doing SLR1 when we have LALR.
      Modify table production to handle LALR.
      Should 'check look ahead' save LALR lookaheads.
      Selectively avoid creating bit tables.
   #
# Temporary measures #                                    PR page PR

                                                          PR page PR
INT left margin := 1,
INT place := 0;
```



```
INT right margin = 130;
[1 : right margin] CHAR print line;

PROC indent = VOID : left margin +:= 3;
PROC dedent = VOID :
   IF left margin > 3
   THEN left margin -:= 3 FI;

PROC end line = VOID:
   BEGIN
   print((print line[1 : place], newline));
   place :=
      ( left margin > right margin % 2
      | right margin % 2
      | left margin ) - 1;
   FOR i FROM 1 TO place DO print line[i] := " " OD
   END,

PROC my end line = VOID:
   BEGIN
   left margin +:= 6;
   end line;
   left margin -:= 6
   END;

OP -< = (CHAR c) VOID :
   BEGIN
   IF place >= right margin THEN end line FI;
   print line[place +:= 1] := c
   END # of 'print' characters # ;

OP -< = (REF STRING s) VOID :
   IF
      IF place >= right margin THEN my end line FI;
      INT ub = UPB s;
      INT np = place + ub;
      np <= right margin
   THEN # normal case #
      print line[place+1 : np] := s;
      place := np
   ELSE INT break = right margin - place;
      -< s[1:break];
      my end line;
      -< s[break+1 : ]
   FI # end of 'print' for string variables #;

OP +< = (STRING s) VOID :
   IF
      IF place >= right margin THEN my end line FI;
      INT ub = UPB s;
      INT np = place + ub;
```

```
      np <= right margin
   THEN # normal case #
      print line[place+1 : np] := s;
      place := np
   ELSE LOC FLEX [1 : ub] CHAR t;
      t := s;
      -< t
   FI;

OP -< = (INT i) VOID : +< whole(i, 0),

OP -< = (BOOL b) VOID:
      (b | +< "true" | +< "false");

PROC time = (STRING s) VOID: # if applicable #
   (
   COMMENT No enquiries ---
   dedent; end line; indent;
   +< s; +< " after ";
   +< fixed(clock, 0, 6); +< " seconds"; end line;
   IF g opt THEN collect garbage FI;
   -< available; +< "words of storage are available after ";
   -< collections; +< " garbage collections ";
   +< "which have collected a total of " ; -< garbage;
   +< " words of garbage and have cost ";
   +< fixed(collect seconds, 0, 6);
   +< " seconds of CPU time. "; end line;
   --- end of COMMENT
   SKIP
   );
BOOL g opt := FALSE;
# SLR(1) parser generator #                                  PR page PR

PROC generate parser =
   (REF FILE input, output,
      BOOL pr opt, pl opt, ple opt, f opt, s opt, t opt,
           c opt, lll opt)
   BOOL:
#
   pr opt:   print r matrix.
   pl opt:   print 1 matrix.
   gt opt:    print 1 nonempty matrix.
   f:   print f matrix.
   s opt:   try SLR(1) processing first.
   t opt:   trace building of FSM.
   c opt:   trace configurations of states.
   lll opt:   perform lll check.
#
BEGIN
# Global changes desired:
   Replace "CONFLIST" everywhere by "PROMLIST"
```



```
#
# Other possible changes:
   The lookaheads are not needed to compute the a matrix and the
   margins, with a few exceptions: production transitions are not
   placed in a, but only in the margins.
   The 'm' matrix and entire FSM list structure can then be discarded
   except for representative back transitions for error messages,
   releasing storage for the bit matrix computations.
   The net effect would be to reduce the main storage required.
#
+< progname; end line;
time("Started");
+< "Options:"; indent; end line;
   +< "pr opt "; -< pr opt; end line;
   +< "pl opt "; -< pl opt; end line;
  +< "ple opt "; -< ple opt; end line;
   +< "LL1 opt "; -< lll opt; end line;
   +< "f opt "; -< f opt; end line;
   +< "s opt "; -< s opt; end line;
   +< "t opt "; -< t opt; end line;
   +< "c opt "; -< c opt; end line;
   +< "g opt "; -< g opt; end line;
   dedent; end line;

   #Modes#                                                   PR page PR

MODE
   STATE = STRUCT(
         REF TRANSITIONLIST out # all transitions leading out of
            this state #,
      REF TRANSITIONLIST in
         # all transitions leading into this state;
         repeatedly following the first 'in' transition of the 'in'
         transitionlist of each state will eventually lead to
         the start state #,
         REF CONFLIST conf # the configurations of this state #,
      BOOL is adequate # initially TRUE. 'is adequate' is assigned
         FALSE only when the state is judged adequate; i.e.,
         all its lookahead is resolved. #,
         REF STATE next # in same hash bucket #,
                     link # 'link' links new states together
                       until they are fully processed #,
         INT number),

      TRANSITIONLIST = STRUCT(
         REF TRANSITION this,
         REF TRANSITIONLIST next),

      TRANSITION = STRUCT(
         REF STATE from,
         MARKER symbol,
```

```
   REF STATE to # production transitions lead nowhere;
      therefore their 'to' fields are NIL #,
   INT number,
   INT scan # used to prevent endless recursion in
      'lalr look ahead' #
   ),

PRODUCTION = STRUCT(
   SYMBOL left,
   PROMOTION right,
   INT number,
   BOOL useful),

PRODUCTIONLIST = STRUCT(
   PRODUCTION this,
   REF PRODUCTIONLIST next),

PROD = PRODUCTION,

SYMBOL = REF SYM,

SYM = STRUCT(
   STRING name,
   INT number,
   BOOL isterminal,
   REF CONFLIST attachment,
   REF STATE states # states are hashed according to
      access symbol; STATES points to the hash bucket #,
   BOOL useful, productive, empty,
   REF PRODUCTIONLIST definitions),

SYMBOLLIST = STRUCT(
   SYMBOL this,
   REF SYMBOLLIST next),

CONFIGURATION = STRUCT(
   SYMBOL sym,
   PROMOTION promote),

CONF = CONFIGURATION,

CONFLIST = STRUCT(
   PROMOTION this,
   REF CONFLIST next),

PROMOTION = UNION(REF CONF, REF PROD),

MARKER = UNION(SYMBOL, REF PROD),

GRAMMAR = STRUCT(
   REF [] REF PRODUCTION productions,
```



```
         REF PRODUCTION start production,
         REF [] SYMBOL symbols,
                         terminals,
                         nonterminals,
         SYMBOL start,
                 end of file
         ),
PR page PR

PRIO ORAB = 1,
       ANDAB = 1,
       MIN = 1,
       MAX = 1;

OP ORAB = (REF BOOL a, BOOL b)REF BOOL : a:= a OR b,
   ANDAB = (REF BOOL a, BOOL b) REF BOOL: a:= a AND b,
   ORAB = (REF BITS a, BITS b)REF BITS : a:= a OR b,
   ANDAB = (REF BITS a, BITS b)REF BITS : a:= a AND b,
   MIN = (INT a,b)INT: (a > b| b| a),
   MAX = (INT a,b)INT : (a > b| a| b);

OP = = (UNION(MARKER, PROMOTION) a, b) BOOL :
         NOT (a /= b),
     /= = (UNION(MARKER, PROMOTION) a, b) BOOL :
   CASE a
   IN
      (SYMBOL a): ( b | (SYMBOL b) : a :/=: b | TRUE ),
      (REF PROD a): ( b | (REF PROD b): a :/=: b | TRUE ),
      (REF CONF a): ( b | (REF CONF b): a :/=: b | TRUE )
   OUT error("invalid parameter to /= - Parser generator error");
      GOTO stop
   ESAC;

OP /= =(REF CONFLIST c,d) BOOL:
         NOT (c = d),
      = =(REF CONFLIST c,d) BOOL:
       c <= d AND d <= c,

      <= = (REF CONFLIST c,d) BOOL:
          BEGIN
          REF CONFLIST l,m;
          l:= c;
          WHILE IF l :/=: REF CONFLIST (NIL)
             THEN # test that 'this' OF 'l' is in 'd'. #
                m := d;
                WHILE IF m :/=: REF CONFLIST (NIL)
                   THEN
                      this OF l /= this OF m
                   ELSE FALSE
                   FI
                DO m:= next OF m
```

```
            OD;
            # assert 'm' :=: NIL iff this OF 'l' is not in d #
            m :/=: REF CONFLIST (NIL)
         ELSE FALSE
         FI
      DO l:= next OF l
      OD;
      # assert l :=: NIL iff c is contained in d #
      l :=: REF CONFLIST ( NIL )
      END # of <= #;

OP SIZE = (REF CONFLIST c) INT :
   BEGIN
   INT i := 0;
   LOC REF CONFLIST d := c;
   WHILE d :/=: REF CONFLIST(NIL)
   DO d := next OF d; i +:= 1
   OD;
   i
   END # of 'size' #;

PROC for right side =
   (  REF PRODUCTION p,
      PROC(REF SYMBOL)VOID x )
   VOID:
   BEGIN
   PROMOTION prom := right OF p;
   WHILE CASE prom
      IN (REF PROD p) : FALSE,
         (REF CONF c): (x(sym OF c);
         prom := promote OF c;
            TRUE)
      ESAC
   DO SKIP OD
   END # of 'for right side' #;
# Output #                                                PR page PR

MODE PRINTABLE = UNION( CHAR, STRING, INT, REF SYM,
      PROC(REF FILE) VOID,
      REF PROD, REF PRODUCTIONLIST, REF CONF,
      REF [] REF PROD,
      REF SYMBOLLIST,
      REF TRANSITIONLIST,
      REF STATE, REF TRANSITION, GRAMMAR);

PROC error = ([]PRINTABLE message) VOID :
   BEGIN
   end line; +< "Error detected: ";
   show(message);
   end line
   END #error#;
```



```
PROC sys error = ([]PRINTABLE message) VOID :
   BEGIN
   end line; +< "System error detected: ";
   show(message);
   end line
   END # of 'sys error' #;
PROC([]PRINTABLE) VOID sys err = sys error;

PROC show = ([] PRINTABLE x) VOID:
   FOR i FROM LWB x TO UPB x
   DO
   CASE x[i]
   IN
      (PROC (REF FILE) VOID x) : (end line; x(stand out)),
      (CHAR c): -< c,
      (STRING s): +< s,
      (INT i): -< i,
      (REF SYM s): -< i,
      (REF SYMBOLLIST s): -< s,
      (REF PRODUCTION p): -< p,
      (REF PRODUCTIONLIST p): -< p,
      (REF [ ] REF PROD p): -< p,
      (REF CONFIGURATION c): -< c,
      (REF STATE s): -< s,
      (REF TRANSITION t): -< t,
      (REF TRANSITIONLIST t): -< t,
      (GRAMMAR g): -< g
   OUT +< "Unprintable stuff"
   ESAC
   OD # end of 'show' #;

OP -< = (REF SYM s) VOID :
                  IF s :/=: REF SYM(NIL)
                   THEN -< name OF s
                   ELSE +< "*ref sym nil*"
                  FI,

OP -< = (REF SYMBOLLIST s) VOID :
                           IF s :=: REF SYMBOLLIST(NIL)
                           THEN SKIP
                           ELSE -< this OF s; +< ", ";
                                  -< next OF s
                           FI,

OP -< = (REF PRODUCTIONLIST p) VOID:
      (p :=: NIL | SKIP | -< this OF p; -< next OF p),

OP -< = (REF [ ] REF PROD p) VOID:
      IF p :=: REF [] REF PROD(NIL)
      THEN SKIP
      ELSE
```

```
      FOR i FROM LWB p TO UPB p
      DO -< p[i]; end line
      OD
   FI,


OP -< = (PROMOTION prom) VOID:
   CASE prom
   IN
   (REF CONFIGURATION c):
      IF c :=: REF CONFIGURATION(NIL)
      THEN +< "* ref conf nil *"
      ELSE
         PROMOTION p := c; PRODUCTION pr;
         WHILE CASE p
            IN
            (REF CONF c): (p := promote OF c; TRUE),
            (REF PROD p): (pr := p; FALSE)
            ESAC
         DO SKIP OD ;
         show configuration(pr, c)
      FI,
   (REF PRODUCTION p):
         show configuration(p, REF PROD(NIL))
   OUT syserr("+< promotion fails")
   ESAC,

OP -< = (REF CONFLIST c) VOID:
   IF c :=: REF CONFLIST(NIL)
      THEN +< "* ref conflist nil *"
   ELSE REF CONFLIST l := c;
      WHILE l :/=: REF CONFLIST(NIL)
      DO -< this OF l; end line;
         l := next OF l
      OD
   FI,

OP -< = (REF STATE s) VOID:
      (+< "state number ", -< number OF s),

OP -< = (REF TRANSITION t) VOID :
      ( +< "transition from "; -< from OF t;
         +< " to ";
         IF to OF t :=: REF STATE(NIL)
         THEN +< "nowhere"
         ELSE -< to OF t
         FI;
         +< " under ";
         CASE symbol OF t
         IN
            (SYMBOL s): -< s,
```



```
            (REF PROD p): -< p
         OUT +< "????"
         ESAC
      ),

OP -< = (REF TRANSITIONLIST t) VOID:
   IF t :/=: REF TRANSITIONLIST(NIL)
   THEN -< this OF t; +< ", "; -< next OF t
   FI,

OP -< = (GRAMMAR g) VOID:
   BEGIN print(newpage); -< productions OF g END,

PROC show configuration = (REF PROD p, PROMOTION c) VOID:
   BEGIN
   -< number OF p; +< ": "; -< left OF p;
   CHAR sep := ":";
   PROMOTION pro := right OF p;
   WHILE
      IF c = pro THEN +< " ..." FI;
      CASE pro
      IN (REF PROD): FALSE,
         (REF CONF c): ( -< sep; -< " "; -< sym OF c;
                         pro := promote OF c;
                         TRUE)
      ESAC
   DO sep := ","
   OD;
      IF sep = ":" THEN +< ": " FI;
   +< ". "
   END # of 'show configuration' #;

OP +< = (REF STATE s) VOID :
         IF s:/=: NIL
         THEN IF in OF s :/=: REF TRANSITIONLIST (NIL)
            THEN REF TRANSITION t = this OF in OF s;
               IF t :/=: REF TRANSITION(NIL)
               THEN +< from OF t;
                  CASE symbol OF t
                  IN (SYMBOL s): ( -< " "; -< name OF s)
                  OUT syserr(("nonsymbol on transition ", t))
                  ESAC
               FI
            FI
         FI;

#Reading grammars #                                          PR page PR

#
   Grammars are read in according to the following
      grammar:
```

```
   grammar: direction, productions, ".".
   productions: production, ".";
                productions, ".", production.
   direction: empty, """forward""", """backward""".
   production: non-terminal, ":", right sides.
   right side: empty;
               symbol;
               symbol, ",", right side.
   right sides: right side;
                right side; ";"; right sides.
   symbol: terminal; nonterminal.
   nonterminal: TAG.
   terminal: strict terminal; pseudo terminal.
   x:: strict; pseudo.
   x terminal: x mark, x images, x mark.
   x images: CHARACTER;
             x mark, xmark;
             character, x images;
             x mark, x mark, x images.
   strict mark: """".
   pseudo mark: "'".
#

PROC read grammar = (REF GRAMMAR g)BOOL:

   BEGIN

   BOOL input line ended := FALSE;
   CHAR input state := " ";

   PROC char in string=
      (CHAR c, STRING s)BOOL:
      BEGIN BOOL val:= FALSE;
      FOR i FROM LWB s TO UPB s
      WHILE NOT ( val := c = s[i])
      DO SKIP OD;
      val
      END # char in string#;

   PROC is letter = (CHAR c)BOOL:
          c = "<" OR c = ">" OR
         (c >= "a")AND (c <= "z"),
      is letdig = (CHAR c)BOOL:
          c = "<" OR c = ">" OR
           c >= "a" AND c <= "z"
      OR c >= "0" AND c <= "9";

   CHAR char:= " ";

   STRING line := "", INT linept := 1;
```



```
   PROC next ch = BOOL:
      BEGIN
      input line ended := FALSE;
      WHILE linept > UPB line
      DO get(input, (newline, line));
         print((input state, " ", line, newline));
         linept := 1;
         input line ended := TRUE
      OD;
      char := line[linept];
      linept +:= 1;
      TRUE
      END;

   PROC skip comments = VOID:
         WHILE char = "#" OR char = "[" OR char = "+"
         DO
         IF char = "["
         THEN input state := "]";
            WHILE nextch; char /= "]"
            DO SKIP OD;
            input state := " ";
            nextch
         ELSE
            CHAR ch = char;
            WHILE nextch;
               IF char = ch
               THEN nextch; FALSE
               ELIF input line ended
               THEN error(("unfinished comment")); FALSE
               ELSE TRUE
               FI
            DO SKIP OD
         FI
         OD # end of 'skip comments' # ;

   PROC next char = BOOL:
         BEGIN
         next ch;
         skip comments;
         TRUE
         END # of next char #;

   PROC coast = VOID:
      IF char = " "
      THEN WHILE next char; char = " " DO SKIP OD
      FI;

   PROC verslind= (STRING stop)VOID:
      BEGIN input state := "-";
      WHILE NOT char in string(char, stop)
```

```
   DO next char
   OD;
   input state := " "
   END;

PROC look up terminal =
   (STRING name) SYMBOL:
      look up symbol (terminals, name, nmb terminals),

PROC look up nonterminal=
   (STRING name) SYMBOL:
      look up symbol(nonterminals, name, nmb nonterminals),

PROC look up symbol=
   (REF REF SYMBOLLIST table,
      STRING name,
      REF INT counter) SYMBOL :
      BEGIN
      REF SYMBOLLIST t := table;
      WHILE IF t :/=: REF SYMBOLLIST(NIL)
              THEN name OF this OF t /= name
              ELSE FALSE
              FI
      DO t:= next OF t
      OD;
      IF t :=: REF SYMBOLLIST(NIL)
      THEN t := table := HEAP SYMBOLLIST :=
         (HEAP SYM :=
            (name, counter +:= 1, SKIP, NIL, NIL,
               FALSE, FALSE, FALSE, NIL
            ),
         table
         )
      FI;
      this OF t
      END;

PROC read nonterminal = (REF STRING n) BOOL:
   IF coast; isletter(char)
   THEN
      WHILE isletdig(char)
      DO n +:= char;
         next char;
         IF char = " "
         THEN coast;
            IF isletdig(char)
            THEN n +:= " "
            FI
         FI
      OD;
      TRUE
```



```
      ELIF char = "/"
      THEN n +:= "/"; nextchar;
         IF read terminal(n)
         THEN TRUE
         ELSE read nonterminal(n)
         FI
      ELSE FALSE
      FI # end of read nonterminal #;

   PROC in nonterminal=(REF SYMBOL s) BOOL:
      IF STRING n := ""; read nonterminal(n)
      THEN s := look up nonterminal(n);
         IF (n[1] = "/" OR n[1] = "<" OR n[1] = ">")
            AND NOT empty OF s
         THEN productionlist :=
               HEAP PRODUCTIONLIST :=
                  ((s, SKIP, nmb prod +:= 1, FALSE),
                     productionlist);
            right OF this OF productionlist := this OF
                  productionlist;
            empty OF s := TRUE
         FI;
         TRUE
      ELSE FALSE
      FI # end of 'in nonterminal' # ;

   PROC read terminal = (REF STRING n) BOOL:
      IF coast; char = "'" OR char = """"
      THEN CHAR x = char; n +:= x;
         input state := char;
         WHILE
            next ch;
            IF char = x
            THEN n +:= x; next ch; char = x
            ELIF input line ended
            THEN error(("unfinished terminal ", n)); FALSE
            ELSE TRUE
            FI
         DO n +:= char
         OD;
         input state := " ";
         skip comments;
         TRUE
      ELSE FALSE
      FI # end of 'read terminal' # ;

   PROC in terminal = (REF SYMBOL s) BOOL:
      IF STRING n := ""; read terminal(n)
      THEN
         IF UPB n > 2
         THEN s:= look up terminal(n); TRUE
```

```
      ELSE error("empty terminal"); FALSE
      FI
   ELSE FALSE
   FI # end of in terminal#;

PROC in symbol = (REF SYMBOL s) BOOL:
   (in nonterminal(s) | TRUE| in terminal(s));

PROC in right tail = ( REF PROMOTION c,
                       REF PRODUCTION p) BOOL:
   IF SYMBOL s;
      in symbol(s)
   THEN
   IF forward
   THEN c := HEAP CONFIGURATION :=
      (  s,
         IF coast; char=","
         THEN IF PROMOTION c;
               next char;
               in right tail(c,p)
            THEN c
            ELSE p
            FI
         ELIF char = "." OR char = ";"
         THEN p # empty alternative #
         ELSE error(
               ""","", "".."", or "";"" expected but not found");
            p
         FI
      );
   TRUE
   ELSE c := HEAP CONFIGURATION := (s, p);
         WHILE coast; char = ","
         DO IF next char; in symbol(s)
            THEN c := HEAP CONFIGURATION := (s, c)
            ELSE error("missing or invalid symbol");
               verslind(",;:.")
            FI
         OD;
         TRUE
   FI
   ELSE c := p; TRUE
   FI # end of in right tail#;

PROC in right side = (REF PROD p) BOOL:
         # yes, only one REF here #
   IF
      HEAP PROMOTION c;
      in right tail (c, p)
   THEN right OF p := c;
      TRUE
```



```
      ELSE FALSE
      FI;

   PROC in production = (REF REF PRODUCTIONLIST l) BOOL:
      IF
         SYMBOL left; in nonterminal(left)
      THEN IF coast; char=":"
         THEN
            WHILE char = ":" OR char = ";"
            DO
               next char;
               l := HEAP PRODUCTIONLIST :=
                  ((left, SKIP, nmb prod +:= 1, FALSE), l);
               IF in right side(this OF l)
               THEN coast
               ELSE error("invalid right side.")
               FI
            OD;
            IF char /= "."
            THEN error("invalid right side terminator");
               verslind(".");
               FALSE
            ELSE TRUE
            FI
         ELSE error("no "":"""); verslind(":"); FALSE
         FI
      ELSE error("no nonterminal on left");
         FALSE
      FI # end of in production #;

   PROC in grammar= BOOL:
      BEGIN
      (STRING s; get(input, s));
      STRING direction := "";
      forward :=
         IF NOT read terminal(direction)
         THEN TRUE
         ELIF direction = """forward""" THEN TRUE
         ELIF direction = """backward""" THEN FALSE
         ELSE error(("invalid direction", direction,
               ". ""forward"" is assumed. "));
            TRUE
         FI;
      WHILE coast; char /= "."
      DO IF in production (production list)
         THEN IF program symbol :=: SYMBOL(NIL)
            THEN program symbol:= left OF this OF production list
            FI
         ELSE error("invalid production");
            verslind(".")
         FI;
```

```
      next char
   OD;
   TRUE
   END # of in grammar # ;
```

PR page PR

```
BOOL forward := TRUE;
INT nmb prod:= 0,
   nmb terminals:= 0,
   nmb nonterminals:= 0;
REF PRODUCTIONLIST productionlist := NIL;
REF SYMBOLLIST terminals := NIL,
      nonterminals := NIL;
SYMBOL program symbol := NIL;
SYMBOL start symbol = look up nonterminal ("* start *"),
      end of file = look up terminal ( "* end of file *" );

in grammar;

production list := HEAP PRODUCTIONLIST :=
      (SKIP, production list);
REF PRODUCTION start production = this OF production list;
start production :=
      (start symbol,
         HEAP CONFIGURATION:= (
            IF program symbol :=: SYMBOL(NIL)
            THEN error("grammar has no productions"); start symbol
            ELSE program symbol
            FI,
            HEAP CONFIGURATION:= (
               end of file,
               start production
               )
            ),
      nmb prod +:= 1,
      FALSE
      );

HEAP [ 1 : nmb prod] REF PRODUCTION productions;
HEAP [1: nmb terminals + nmb nonterminals]
      SYMBOL symbols;

WHILE production list :/=: REF PRODUCTIONLIST ( NIL )
DO
   REF PRODUCTIONLIST here = production list;
   production list := next OF here;
   productions[number OF this OF here] := this OF here;
   next OF here := definitions OF left OF this OF here;
   definitions OF left OF this OF here := here
OD;
```



```
   WHILE terminals :/=: REF SYMBOLLIST(NIL)
   DO
      symbols[number OF this OF terminals] := this OF terminals;
      is terminal OF this OF terminals:= TRUE;
      terminals:= next OF terminals
   OD;

   WHILE nonterminals :/=: REF SYMBOLLIST (NIL)
   DO
      symbols[number OF this OF nonterminals +:= nmb terminals] :=
            this OF nonterminals;
      is terminal OF this OF nonterminals := FALSE;
      nonterminals:= next OF nonterminals
   OD;

   g :=
      (productions, start production,
      symbols, symbols[1:nmb terminals],
            symbols[nmb terminals +1: nmb terminals + nmb nonterminals
                  @ nmb terminals + 1],
      start symbol, end of file);

   TRUE

   END # of read grammar#;
```

PR page PR

```
GRAMMAR g;

time("Read grammar");
IF read grammar(g)
THEN

time("Grammar read");


PROC extract production data = VOID:
   BEGIN INT l; PROMOTION p;
   FOR i FROM 1 TO nmb productions
   DO target[i] := number OF left OF production[i];
      production length[i] := (l := 0; p := right OF production [i];
         FOR i FROM 0
         WHILE CASE p
            IN (REF PROD) : (l := i; FALSE),
            (REF CONF c) : (p := promote OF c; TRUE)
            ESAC
         DO SKIP OD;
         l
         )
   OD
   END # of extract production data #;
```

```
ROC find empty and useless nonterminals = VOID :
  BEGIN

  FOR i TO UPB symbol
  DO empty OF symbol[i] := FALSE;
     productive OF symbol[i] := FALSE;
     useful OF symbol[i] := FALSE
  OD;

  FOR i FROM LWB terminal TO UPB terminal
  DO productive OF symbol[i] := TRUE
  OD;

  WHILE
     BEGIN
     BOOL change := FALSE;
     PRIO NEW= 1;

     OP NEW=(REF BOOL d,BOOL s) VOID:
        ( NOT d AND s
        | d := TRUE; change := TRUE
        );

     FOR pn FROM 1 TO UPB production
     DO REF PRODUCTION p= production [pn];
        PROMOTION r := right OF p;
        BOOL emptyright:= TRUE,
           productive right:= TRUE;

        WHILE
           CASE r
           IN (REF CONFIGURATION c):
                 BEGIN
                 emptyright ANDAB empty OF sym OF c;
                 productiveright ANDAB productive OF sym OF c;
                 r := promote OF c;
                 TRUE
              END
           OUT FALSE
           ESAC
        DO SKIP OD;

        SYMBOL left = left OF p;
        empty OF left NEW empty right;
        productive OF left NEW productiveright;

        IF productive right
           AND ( useful OF left
              OR ( left :=: start symbol)
              )
           AND # for efficiency only#
```



```
                 NOT useful OF p
              THEN useful OF p:= TRUE;
                 useful OF left := TRUE;
                 r:= right OF p;
                 WHILE
                    CASE r
                    IN (REF CONFIGURATION c):
                       BEGIN
                          useful OF sym OF c NEW TRUE;
                          r := promote OF c;
                          TRUE
                       END
                    OUT FALSE
                    ESAC
                 DO SKIP OD
              FI

        OD;

        change
        END
     DO SKIP OD

     END # of find empty...#;

  PROC print symbols= VOID:
     BEGIN

     PROC s = (SYMBOL s) VOID :
        BEGIN
        end line;
        IF useful OF s THEN +< "     " ELSE +< ">>>>>" FI;
        -< " ";
        IF empty OF s THEN +< "empty" ELSE +< "     " FI;
        -< " ";
        IF NOT productive OF s THEN +< "not productive"
              ELSE +< "              " FI;
        IF NOT useful OF s THEN +< " not useful "
              ELSE +< "            " FI;
        -< name OF s
        END;

     end line; print(newpage); +< "Terminals";
     FOR i FROM LWB terminal TO UPB terminal
     DO s( terminal[i] )
     OD;
     end line; +< "Nonterminals";
     FOR i FROM LWB nonterminal TO UPB nonterminal
     DO s(nonterminal[i])
     OD;
     end line
```

```
   END # of 'print symbols' #;

# Bit matrices #                                        PR page PR

PROC bit = (REF [,]BITS m, INT i, j) BOOL :
   BEGIN
   INT iw = i OVER bits width,
         ib = i MOD bits width;
   (ib + 1) ELEM m[iw, j]
   END # of bit #;

PROC setbit = (REF [,] BITS m, INT i, j) VOID:
   BEGIN
   INT iw = i OVER bits width,
         ib = i MOD bits width;
   REF BITS e = m[iw, j];
   e := e OR 2 r 1 SHL (bits width - ib - 1)
   END # of set bit # ;

PROC print bit matrix =
   (REF [,] BITS a,
      INT l1, u1, l2, u2,
      CHAR mark) VOID:
   BEGIN
   time("Print bit matrix");
   FOR p FROM l1 BY 50 TO u1
   DO INT q = u1 MIN p + 49;
      FOR r FROM l2 BY 50 TO u2
      DO INT s = u2 MIN r + 49;

         end line; print(newpage);

         # Heading of 8 characters per symbol #
         FOR z FROM 1 TO 8
         DO FOR u FROM r TO s
            DO IF UPB name OF (symbol[u]) < z
               THEN -< " " ELSE -< (name OF (symbol[u])) [z]
               FI
            OD;
            end line
         OD # end of heading #;

         FOR t FROM p TO q
         DO STRING background =
                  ( t MOD 5 = 0 | "-+" | " !" ) [AT 0];
            FOR u FROM r TO s
            DO IF bit(a, t, u)
               THEN -< mark
               ELSE -< background[ABS(u MOD 5 = 0)]
               FI
            OD;
```



```
               -< t; -< " "; -< name OF symbol[t]; end line
            OD

         OD
      OD
      END # of print matrix #;
   # Create bit arrays #                                  PR page PR
   PROC create bit arrays = (REF REF [,] BITS rpl) VOID :
      BEGIN
      [lownt32 : highnt32, 1 : nsymbols] BITS pr,
      HEAP [lownt32 : highnt32, 1 : nsymbols] BITS pl,
      [ 0 : nsymbols 32, 1 : nsymbols ] BITS adj, temp;

      #  Compute 'pl', 'pr', and 'adj'  #

         FOR i FROM 1 LWB pr TO 1 UPB pr
         DO FOR j FROM 2 LWB pr TO 2 UPB pr
            DO pr[i, j] := pl[i, j] := plnonempty[i, j] := 2 r 0
            OD
         OD;

         FOR i FROM 1 LWB adj TO 1 UPB adj
         DO FOR j FROM 2 LWB adj TO 2 UPB adj
            DO adj[i, j] := temp[i, j] := f[i, j] := 2 r 0
            OD
         OD;

         FOR pi TO UPB production
         DO REF PROD p = production[pi];
         IF NOT useful OF p
         THEN error(("production ", p, " is not useful. "))
         ELSE
            SYMBOL l= left OF p;
            PROMOTION r= right OF p;

            CASE r
            IN (REF PROD): SKIP,
               (REF CONF r):
                  setbit(plnonempty, number OF l, number OF sym OF r)
            ESAC;

            PROMOTION tail:= r,
                  prev := p;

            WHILE
               BEGIN

                  PROMOTION tailtail := tail;

                  WHILE
                     CASE tailtail
```

```
            IN

            (REF PROD tt):
               (CASE prev
                 IN (REF PROD prev): SKIP,
                     (REF CONF prev):
                              setbit(pr, number OF l,
                                number OF sym OF prev)
               ESAC;
               FALSE
               ),

               (REF CONF tt):
                  (CASE prev
                  IN (REF PROD prev):
                     setbit(pl, number OF l,
                        number OF sym OF tt),
                     (REF CONF prev):
                             setbit(adj,
                                number OF sym OF prev,
                                number OF sym OF tt)
                  ESAC;
                  tailtail := promote OF tt;
                  empty OF sym OF tt
                  )

               ESAC
            DO SKIP OD;
            prev := tail;
         CASE tail

         IN (REF PROD): FALSE,

         (REF CONF t):
            BEGIN
            tail:= promote OF t;
            TRUE
            END

         ESAC
         END
      DO SKIP OD
   FI
   OD;

OMMENT The following lines have been commented out.
  They may nonetheless be useful for debugging changes later.
  print bit matrix(pr, lownt, highnt, l, nsymbols, "q");
  print bit matrix(pl, lownt, highnt, l, nsymbols, "k");
  print bit matrix(pl nonempty, lownt, highnt, l, nsymbols, ">");
  print bit matrix(adj, l, nsymbols, l, nsymbols, "=");
```



```
COMMENT
   time("Compute closures");

   # At this point,
      pl[i, j] iff i => xxxj... and xxx =>* empty
      pr[i, j] iff i => ...jxxx and xxx =>* empty
      adj[i,j] iff x => ...ixxxj... and xxx =>* empty,
      plnonempty[i, j] iff i => j...
   Now compute the symmetric transitive closures of
         pl,pr, and plnonempty;
   #
# Bit matrices: transitive closures #                          PR page PR

   PROC close= (REF [,] BITS m) VOID:
      # replace m by its transitive closure #
      FOR j FROM lownt TO highnt
      DO FOR k FROM l TO nsymbols
         DO IF bit (m,j,k)
            THEN FOR i FROM lownt 32 TO highnt 32
               DO m[i,k] ORAB m[i,j]
               OD
            FI
         OD
      OD;

   close (pl);
   IF s opt OR pr opt THEN close(pr) FI;
      # otherwise, 'pr' is not needed. #
   close (plnonempty);
   FOR i FROM lownt TO highnt
   DO setbit (pr,i,i);
      setbit (pl,i,i);
      setbit (plnonempty,i,i)
   OD;

   # At this point,
      pl[i, j] iff i =>* j...
      pr[i, j] iff i =>* ...j
      adj[i,j] iff x => ...ixxxj... and xxx =>* empty,
      plnonempty[i, j] iff i => j... using no empty
         productions
   #

# Bit matrices: the follows matrix #                           PR page PR

   IF s opt OR f opt
   THEN
   time("Compute ''f''");
   PRIO /* = 7;

   OP /* = (REF [,] BITS a, b) REF [,] BITS :
```

```
# a transpose times b.
   'b' contains only one part of the 'b' that is to be
   multiplied. It diagonal is extended with ones, thus:
     10000000  <-  1
     01000000
     00100000
     bbbbbbbb  <-  lownt
     bbbbbbbb
     bbbbbbbb
     bbbbbbbb
     bbbbbbbb  <-  nsymbols = highnt

let 'u' be the [,]BOOL which is packed as [,]BITS
in 't'. 'at' is the transpose of 'a' .
then:

u[i, j] = OR[k] at[i, k] AND b[k, j]

   = OR[k] (a[k, i] AND
      IF k < lownt THEN k = j ELSE b[k, j] FI )

   =      ( OR[k < lownt] a[k, i] AND (k = j) )
      OR ( OR[k >= lownt] (a[k, i] AND b[k, j] ) )

   =      IF j < lownt THEN a[j, i] ELSE FALSE FI
     OR OR[k >= lownt] (a[k, i] AND b[k, j])

#

BEGIN HEAP [0 : n symbols 32, 1 : n symbols] BITS t;
   BITS 1;

   FOR i FROM 1 LWB t TO 1 UPB t
   DO FOR j FROM 2 LWB t TO 2 UPB t
      DO t[i, j] := 2 r 0
      OD
   OD;

   FOR i FROM 1 TO nsymbols
   DO
      FOR j FROM 1 TO n symbols
      DO
         IF
            IF
               (j < lownt | bit(a, j, i) | FALSE )
            THEN TRUE
            ELSE
               1 := 2 r 0;
               FOR k FROM lownt 32 TO high nt 32
               DO 1 ORAB (a[k, i] AND b[k, j])
               OD;
```



```
                  1 /= 2 r 0
               FI
            THEN setbit(t, i, j)
            FI
         OD
      OD;
      t
   END;

   f := (adj /* pr) /* pl
      #   pr t * a * pl = (a t * pr) t * pl
         notice that the diagonals of 'pr' and 'pl' must be
         extended with ones.
      #
   FI # end of 'f' processing #;

   IF pr opt
   THEN print bit matrix(pr, low nt, high nt, 1, n symbols, "r")
   FI;
   IF pl opt
   THEN print bit matrix(pl, low nt, high nt, 1, nsymbols, "1")
   FI;
   IF ple opt
      OR TRUE
   THEN print bit matrix(pl nonempty, low nt, high nt,
            1, n symbols, ">")
   FI;
   IF f opt
   THEN print bit matrix(f, 1, n symbols, 1, n symbols, "f")
   FI;

   rpl := pl

   END # create bit matrices #;

PROC destroy bit matrices = VOID :
   BEGIN
   f := NIL;
   plnonempty := NIL
   # ; collect garbage #
   END # of destroy bit matrices# ;
                                                        PR page Pl
# Elementary operations on states, configurations, and transitions #

PROC makestate=
      (REF CONFLIST c,
         SYMBOL access) REF STATE:
   BEGIN

   # hashing on access #
   REF STATE thesestates :=
```

```
      IF access :=: NIL
      THEN NIL # only the start state may have no access #
      ELSE states OF access
      FI;

   # search for an equivalent state#
   REF STATE s:= these states;
   WHILE
      IF s :/=: REF STATE (NIL)
      THEN conf OF s /= c
      ELSE FALSE
      FI
   DO s:= next OF s
   OD;

   IF s:=: REF STATE (NIL)
   THEN # new state #
      s := newstates := thesestates :=
         HEAP STATE :=
            (NIL, NIL, c,
               FALSE, these states, new states, nmb states +:= 1);
      IF t opt THEN +< "new "; -< s; end line FI;
      IF c opt THEN indent; -< s; end line; -< conf OF s; dedent;
            end line FI
   FI;

   IF access :=: SYMBOL(NIL)
   THEN startstate:= thesestates
   ELSE states OF access := thesestates
   FI;
   s
   END # of makestate # ;

#attach#
PROC attach= (PROMOTION c, SYMBOL s) VOID:
   # Attach the promotion 'p' to the symbol 's', unless it is there
      already. #
   BEGIN
   REF CONFLIST cl:= attachment OF s;
   WHILE IF cl :/=: REF CONFLIST( NIL )
         THEN this OF cl /= c
      ELSE FALSE
      FI
   DO cl:= next OF cl
   OD;
   IF REF CONFLIST (cl) :=: NIL
   THEN attachment OF s:= HEAP CONFLIST:= (c,attachment OF s)
   FI
   END#attach#;

PROC maketransition=
```



```
   (REF STATE from,
      MARKER s,
      REF STATE to
   )VOID:
   # Make a transition from one state to another, unless it is there
      already. 'to' may be NIL, but 'from' may not. #
      BEGIN
      REF TRANSITIONLIST t := out OF from;
      WHILE IF t :/=: REF TRANSITIONLIST(NIL)
         THEN (from OF this OF t :/=: from)
            OR (symbol OF this OF t /= s)
            OR (to OF this OF t :/=: to)
         ELSE FALSE
         FI
      DO t := next OF t
      OD;
      IF t :=: REF TRANSITIONLIST(NIL)
      THEN nmb transitions +:= 1;
         REF TRANSITION new := HEAP TRANSITION
               := (from, s, to, nmb transitions, 0);
         IF t opt THEN +< "new "; -< new; end line FI;
         out OF from := HEAP TRANSITIONLIST :=
               (new, out OF from);
         IF to :/=: REF STATE (NIL)
         THEN
            REF REF TRANSITIONLIST inplace =
               IF in OF to :=: REF TRANSITIONLIST(NIL)
               THEN in OF to
               ELSE next OF in OF to
               FI;
            inplace := HEAP TRANSITIONLIST := (new, in place)
         FI
      FI
      END # of make transition # ;

# FSM states#

PROC for all states = (PROC(REF STATE) VOID x) VOID:
   BEGIN
   IF startstate :/=: REF STATE(NIL)
   THEN x(startstate)
   FI;
   FOR i TO UPB symbol
   DO SYMBOL ac = symbol[i];
      REF STATE st := states OF ac;
      WHILE REF STATE (st):/=: NIL
      DO x(st);
         st:= next OF st
      OD
   OD
   END # for all states #;
```

```
¢ Finite state machine construction #                    PR page PR

¢grow fsm#

PROC grow fsm = VOID :
   BEGIN
   nmb states := nmb transitions := 0;
   newstates:= NIL;
   REF STATE current state := start state :=
      IF useful OF start production
      THEN makestate(
            HEAP CONFLIST:=(right OF startproduction, NIL),
            NIL
            )
      ELSE NIL
      FI;
   newstates := currentstate;
   WHILE newstates :/=: REF STATE(NIL)
   DO # Process a new state : #
      currentstate:= newstates;
      newstates := link OF newstates;
      # Process 'current state': #
      REF CONFLIST cl:= conf OF currentstate;
      WHILE cl :/=: REF CONFLIST (NIL)
      DO # Process configuration 'c': #
         PROMOTION c := this OF cl;
         CASE c
         IN

         (REF PRODUCTION c):
            maketransition(currentstate, c, NIL),

         (REF CONFIGURATION c):
               BEGIN # Promotion transitions are wanted, but
                       we must sort them by symbol.
                       The set of resulting new configurations for
                       each symbol will later become a state.
                       The promotion transitions under a symbol are
                       hung on its 'attach' field.
                     #
               SYMBOL s = sym OF c;
               INT sn = number OF s;
               attach(promote OF c, s);
               IF NOT is terminal OF s
               THEN # A nonterminal: predict any new configurations,
                      and promote them as well. #

                  FOR tn TO UPB symbol
                  DO SYMBOL t = symbol[tn];

                     IF bit (plnonempty, sn, tn)
```



```
                     THEN #assert  s =>* t without using empty
                        productions #
                        REF PRODUCTIONLIST pl := definitions OF t;
                        WHILE pl :/=: REF PRODUCTIONLIST(NIL)
                        DO REF PROD p = this OF pl;
                           IF NOT useful OF p
                           THEN SKIP
                           ELSE
                           CASE right OF p
                           IN (REF PRODUCTION):
                                 make transition (
                                    currentstate, p, NIL),
                             (REF CONF rp):
                                    attach(promote OF rp,
                                          sym OF rp)
                           ESAC
                           FI;
                           pl := next OF pl
                        OD
                     FI
                  OD
               FI
               END

         ESAC;
         cl := next OF cl
      OD;

      #Possible new states have been considered as sets of
         configurations. Now make them into real states#

      FOR sn TO UPB symbol
      DO SYMBOL s = symbol[sn];
         IF attachment OF s :/=: REF CONFLIST (NIL)
         THEN REF STATE q = make state(attachment OF s, s);
            maketransition(currentstate,s,q)
         FI;
         REF REF CONFLIST(attachment OF s) := NIL
      OD
   OD
   END # growfsm#;
# Check look ahead #                                     PR page PR

PROC lll check = VOID:
   for all states(
      (REF STATE s) VOID:
         IF SIZE(conf OF s) > 1
         THEN +< "An LL(1) violation can be reached by reading";
            indent; end line;
            +< s; dedent; end line
         FI
```

```
    )
  # end of lll check #;

ROC check look ahead = (BOOL slr l processing) BOOL :
  BEGIN

  IF NOT slrl processing
  THEN syserr(("temporary bug: only SLRl tables will be produced",
       " even though we perform LALR lookahead checks."))
  FI;

  PROC inadequacy = (REF STATE s) VOID:
     # complain about an inadequacy #
     BEGIN

     errors := TRUE;
     end line; end line;
     +< "An inadequate state can be reached by reading ";
        indent; end line;
     +< s;
     dedent; end line;
     +< "Possible actions are: ";
     indent;
     REF TRANSITIONLIST t := out OF s;
     WHILE t :/=: REF TRANSITIONLIST(NIL)
     DO CASE symbol OF this OF t
        IN (SYMBOL s):
              IF isterminal OF s
              THEN end line; +< "Read "; -< name OF s
              FI,
           (REF PRODUCTION p):
              BEGIN
              end line; +< "Apply production "; -< number OF p;
              +< " with lookaheads ";
              INT l = number OF left OF p;
              FOR i FROM LWB terminal TO UPB terminal
              DO IF bit(f, l, i)
                 THEN -< " "; -< name OF symbol[i]
                 FI
              OD
              END
        ESAC;
        t := next OF t
     OD;
     dedent; end line
     END # of inadequacy #;

        #Check lookahead resumes#
  [l : UPB terminal] BOOL b, c;
  BOOL errors := FALSE;
  for all states ((REF STATE s) VOID:
```



```
  BEGIN
  FOR i TO UPB b DO b[i] := FALSE OD;
  REF TRANSITIONLIST tl := out OF s;
  WHILE # more list exists and no conflict yet found #
     IF tl :/=: REF TRANSITIONLIST (NIL)
     THEN
        REF TRANSITION t = this OF tl;
        CASE symbol OF t
        IN (SYMBOL u):
              IF isterminal OF u
              THEN IF b[number OF u]
                     THEN FALSE
                 ELSE b[number OF u]:= TRUE;
                      TRUE
                 FI
              FI,
           (REF PRODUCTION p):
            IF slrl processing
            THEN BOOL conflict := FALSE;
              INT l = number OF left OF p;
              FOR i FROM LWB terminal TO UPB terminal
              DO REF BOOL bi = b[i];
                 IF bi
                 THEN ( bit(f, l, i) | conflict := TRUE )
                 ELSE bi := bit(f, l, i)
                 FI
              OD;
              NOT conflict
            ELSE
              [LWB terminal : UPB terminal] BOOL look;
              FOR i FROM LWB look TO UPB look
              DO look[i]  := FALSE
              OD;
              lalr look ahead(t, look);
              BOOL conflict := FALSE;
              INT l = number OF left OF p;
              FOR i FROM LWB terminal TO UPB terminal
              DO REF BOOL bi = b[i];
                 IF bi
                 THEN (look[i] | conflict := TRUE )
                 ELSE bi := look[i]
                 FI
              OD;
              NOT conflict
            FI
        ESAC
     ELSE FALSE
     FI
  DO tl := next OF tl
  OD;
  IF tl :/=: REF TRANSITIONLIST(NIL)
```

```
      THEN inadequacy (s)
      FI
      END
      ) # end of state loop #;

   NOT errors

   END # of check look ahead#;
# LALR lookahead #                                          PR page PR
PROC lalr look ahead = (REF TRANSITION t,
                        REF[ #1 : nmb terminals # ]BOOL look
                       ) VOID:
   # OR the LALR(1) lookahead set for the reduction transition
        't' into 'look' #
   BEGIN
   REF PRODUCTION p =
      CASE symbol OF t
      IN (REF PROD p): p
      OUT syserror("not a reduction"); give up
      ESAC;
   # Notice that 'p' must be useful to have a transition #
   REF SYMBOL left = left OF p;

   PROC backward = (REF TRANSITION t,
         REF SYMBOLLIST l) VOID:
      IF t :=: REF TRANSITION(NIL)
      THEN SKIP
      ELIF REF STATE s = from OF t;
         s :=: REF STATE(NIL)
      THEN SKIP
      ELIF l :=: REF SYMBOLLIST(NIL)
      THEN REF TRANSITIONLIST t := out OF s;
         WHILE t :/=: REF TRANSITIONLIST(NIL)
         DO IF symbol OF this OF t = left
            THEN forward(to OF this OF t)
            ELSE SKIP
            FI;
         t := next OF t
         OD
      ELSE REF TRANSITIONLIST in := in OF s;
         WHILE in :/=: REF TRANSITIONLIST(NIL)
         DO IF symbol OF this OF in = this OF l
            THEN backward(this OF in, next OF l)
            FI;
            in := next OF in
         OD
      FI # end of 'explore' #;

   PROC forward = (REF STATE s) VOID:
      BEGIN
      REF TRANSITIONLIST out := out OF s;
```



```
      WHILE out :/=: REF TRANSITIONLIST(NIL)
      DO REF TRANSITION t = this OF out;
         IF scan OF t >= l THEN SKIP
         ELSE scan OF t := l;
            CASE symbol OF t
            IN
            (SYMBOL s):
               IF is terminal OF s
               THEN look[number OF s] := TRUE
               ELSE
                  FOR i FROM l TO nmb terminals
                  DO look[i] ORAB bit(pl, number OF s, i)
                  OD;
                  IF empty OF s THEN forward(to OF t)
                  FI
               FI,
            (REF PROD):
               lalr look ahead(t, look)
            ESAC;
            scan OF t := 0
         FI;
         out := next OF out
      OD
      END # of 'forward' #;

   REF SYMBOLLIST rb := NIL;
   for right side(p,
      (REF SYMBOL s) VOID: rb := HEAP SYMBOLLIST :=
                                    (s, rb));
   # 'rb' now contains the right side backwards #
   backward(t, rb)
EXIT give up: SKIP
   END # of 'lalr look ahead' #;
# Make tables #                                             PR page PR

PROC make name table = VOID:
   FOR sy TO UPB symbol
   DO name[sy] := name OF symbol[sy]
   OD;

PROC destroy grammar = VOID :
   BEGIN symbol := NIL; terminal := NIL;
   nonterminal := NIL; production := NIL
   END;

PROC make transition table = VOID:
   # fill in 'm'.
      m[state number, symbol number], called 'x' below, indicates
      the action to be performed from the state when the symbol is th
      next input character.
         'x' < 0: apply production  - x .
```

```
      'x' = 0: error.
      'x' > 0: accept the symbol and enter state 'x'.
#
BEGIN
FOR st TO nmb states
DO FOR sy TO nsymbols
   DO m[st,sy]:= 0
   OD
OD;
for all states(
   (REF STATE st) VOID:
   BEGIN
   INT stn = number OF st;
   REF TRANSITIONLIST tr := out OF st;
   WHILE tr :/=: REF TRANSITIONLIST (NIL)
   DO CASE symbol OF this OF tr
      IN

      (REF PRODUCTION pr):
         BEGIN
         INT pn = number OF pr,
            l = number OF left OF pr;
         productionlength [pn]:=
            (INT len:= 0;
            PROMOTION v := right OF pr;
            WHILE
               CASE v
               IN (REF CONF vc):
                     (len +:= l;
                     v:= promote OF vc;
                     TRUE
                     ),
               (REF PRODUCTION): FALSE
               ESAC
            DO SKIP OD;
            len
            );
         FOR i TO nsymbols
         DO IF bit(f,l,i)
            THEN m[stn,i]:= - pn
            FI
         OD
         END,

      (SYMBOL sy):
         m[stn, number OF sy] := number OF to OF this OF tr

      ESAC;
      tr:= next OF tr
   OD
   END
```



```
      ) # end of 'st' loop #
   END # of make transition matrix #;

PROC destroy trees = VOID:
   BEGIN
   FOR i TO nsymbols DO symbol[i] := NIL OD;
   FOR i TO nmb productions DO production[i] := NIL OD;
   startstate := NIL
   END # of destroy trees #;
# Squash matrix #                                          PR page PR
PROC squash matrix = VOID:
   BEGIN

   PROC add margin =
      (REF REF [,] INT margin,
      INT row,
      INT new value) INT:
      BEGIN

      PROC enlarge = VOID :
         BEGIN
         HEAP [ l : l UPB margin,
                  l : l + (2 UPB margin) ] INT nmr;
         nmr[,l:2UPB margin]:= margin;
         FOR i TO l UPB margin DO nmr[i,2 UPB nmr]:=0 OD;
         margin:= nmr
         END #of enlarge # ;

      IF 2 UPB margin = 0 THEN enlarge FI;

      REF[] INT slice= margin[row,];
      INT ii := 0;
      FOR i TO UPB slice
      WHILE
         ii := i;
         INT si = slice[i];
         si /= new value AND si /= 0
      DO SKIP OD;
      # assert slice[ii] = new value
               or slice[ii] = 0
               or ii = UPB slice
      #

      IF INT si = slice[ii];
         si /= new value AND si /= 0
      THEN enlarge; ii +:= l
      FI;
      IF margin[row,ii]= 0
      THEN margin[row, ii] := new value
      FI;
      ii
```

```
   END # of add margin#;

apply margin := HEAP [ 1:nmb states, 1:0 ] INT;
read margin := HEAP [1 : nsymbols, 1 : 0] INT;

FOR st TO nmb states
DO FOR sy TO nsymbols
   DO REF INT mx= m[st,sy];
      IF mx > 0
      THEN mx:= add margin(read margin,sy,mx)
      ELSE mx := - add margin(apply margin, sy, -mx)
      FI
   OD
OD;

INT nmb diff elem = 2 UPB read margin + 2 UPB apply margin + 1;
element size :=
   (INT twon:= 1, n:= 0;
   WHILE twon < nmb diff elem
   DO twon *:= 2; n +:= 1
   OD;
   n);
break := 2 UPB read margin;
# For a packed element 'i' of size 'element size' in the array 'a',
   i = 0 will mean error,
   i <= break will mean (read; go to new state), and
   i > break will mean apply production i - break.
#
elements per word := bits width OVER element size;
first word := (1 * nsymbols + 1) OVER elements per word;
last word := (nmb states * nsymbols + nsymbols)
      OVER elements per word;

HEAP [first word: last word] BITS a;

FOR i FROM LWB a TO UPB a DO a[i] := 2 r 0 OD;
FOR st TO nmb states
DO FOR sy TO nsymbols
   DO INT i= st * nsymbols + sy;
      INT word index = i OVER elements per word;
      INT shift = (i MOD elements per word) * element size;
      BITS element = BIN IF INT mx = m[st, sy]; mx < 0
               THEN break - mx
               ELSE mx
               FI;
      a[word index] ORAB element SHL shift
   OD
OD;
newm := a;  m := NIL
END # of squash matrix #;
```



```
PROC write tables = VOID :
   IF output :/=: REF FILE(NIL)
   THEN
   put bin(output, (nmb symbols, nmb terminals,
      nmb productions, nmb states,
      UPB newm,
      2 UPB apply margin, 2 UPB read margin,
      number OF start state,
      number OF end of file symbol));
   FOR i TO UPB name
   DO put bin(output, UPB name[i]);
      FOR j TO UPB name[i] DO put bin(output, name[i][j]) OD
   OD;
   put bin(output, (newm, apply margin, read margin));
   put bin(output, (target, production length))
   FI # end of 'write tables' # ;
                                                          PR page PF

on line end(stand out,
   (REF FILE f)BOOL : (put(f, (newline, "         ")); TRUE)
   );

-< g;

SYMBOL start symbol = start OF g,
         end of file symbol = end of file OF g;
REF PRODUCTION start production = start production OF g;
REF [] SYMBOL symbol := symbols OF g,
                  terminal := terminals OF g,
                  nonterminal := nonterminals OF g;
REF [] REF PRODUCTION production := productions OF g;

INT nmb symbols = UPB symbol,
   n symbols = UPB symbol,
   nmb terminals = UPB terminal,
   nmb nonterminals = UPB nonterminal - LWB nonterminal + 1,
   nmb productions = UPB production,
   low nt = LWB nonterminal,
   high nt = UPB nonterminal;
INT n symbols 32 = nmb symbols OVER bits width,
   low nt 32 = lownt OVER bits width,
   high nt 32 = highnt OVER bits width;

-< nmb symbols; +< " symbols "; end line;
-< nmb terminals; +< " terminals "; end line;
-< nmb nonterminals; +< " nonterminals "; end line;
-< nmb productions; +< " productions"; end line;

[1 : n symbols] STRING name; make name table;

time("Find empties");
```

```
ind empty and useless nonterminals;

ime("Print symbols");
rint symbols;
ime("Printed");

ime("Bit arrays");
EF [,] BITS f:= HEAP [0:nsymbols 32, 1:nsymbols] BITS,
        plnonempty:= HEAP[lownt 32 : highnt 32,
                  1:nsymbols] BITS;
EF [,] BITS pl;

reate bit arrays(pl);

EF STATE startstate := NIL,
  newstates := NIL;
NT nmb states := 0,
  nmb transitions := 0;

ime("FSM");
rowfsm;
< nmb states; +< " states "; end line;
< nmb transitions; +< " transitions"; end line;
ime("End FSM");

F lll opt THEN lll check FI;

F NOT
  IF s opt
  THEN IF time("First we try SLR(1) processing");
        check look ahead(TRUE)
     THEN TRUE
     ELSE time("SLR(1) fails; try LALR(1)");
        check look ahead(FALSE)
     FI
  ELSE time("LALR lookahead calculation");
     check look ahead(FALSE)
  FI
HEN error("inadequate states cause suppression of parse tables");
     GOTO fail
I;

F output :=: REF FILE(NIL)
HEN GOTO done
I;

1:nmb productions] INT target,
                        production length;
  extract production data;

EF[,] INT m:= HEAP[1:nmb states, 1:nmb symbols] INT;
```



```
time("Transition table");
make transition table;

destroy grammar; # no longer needed #
destroy bit matrices # no longer needed # ;

# Output variables for 'squash matrix' #
REF [] BITS newm;
REF [,] INT apply margin, read margin;
INT element size, elements per word, first word, last word, break;

time("Compress");
squash matrix;

time("Output");
write tables;

done:

   time("Finish");
   TRUE
ELSE error("invalid grammar");
   time("Finish");
   FALSE

FI

EXIT fail:
   time("Finish");
   FALSE
END # of 'generate parser' # ;
                                                              PR page PR
BEGIN

BOOL pr opt, pl opt, ple opt, f opt, s opt, t opt, c opt, lll opt;

PROC rfn = (REF STRING s) BOOL:
   # Read a string denotation from stand in.
      Yield TRUE if a string denotation is found, and assign it
         to 's';  otherwise yield FALSE and assign junk to 's';
   #
   BEGIN CHAR c := " ";
   on logical file end(stand in, (REF FILE f) BOOL: GOTO x);
   s := "";
   WHILE c = " " DO read(c); print(c) OD;
   IF c = "'" OR c = """"
   THEN on logical file end( stand in,
            (REF FILE f) BOOL: GOTO y
         );
      CHAR q = c; read(c); print(c);
      WHILE IF c = q
```

```
      THEN read(c); print(c); c = q
      ELSE TRUE
      FI
   DO s +:= c; read(c); print(c)
   OD;
   TRUE
ELIF c = "."
THEN s := ""; FALSE
ELIF c = "+" OR c = "-"
THEN BOOL val := c = "+";
   WHILE c /= " "
   DO IF c = "p"
      THEN read(c); print(c);
         IF c = "r" THEN pr opt := val
         ELIF c = "l" THEN pl opt := val
         ELSE print("?")
         FI
      ELIF c = ">" THEN ple opt := val
      ELIF c = "l" THEN lll opt := val
      ELIF c = "f" THEN f opt := val
      ELIF c = "g" THEN g opt := val
      ELIF c = "s" THEN s opt := val
      ELIF c = "t" THEN t opt := val # trace transitions #
      ELIF c = "c" THEN c opt := val # trace configurations #
      ELIF c = "+" THEN val := TRUE
      ELIF c = "-" THEN val := FALSE
      ELSE print("?")
      FI;
      read(c); print(c)
   OD;
   rfn(s)
ELSE print("Invalid first character for file name");
   s := "";
   FALSE
FI
EXIT x:
   s := ""; FALSE

EXIT y:
   print("Invalid or otherwise improper file name");
   s := ""; FALSE
END # of 'rfn' #;

ROC find file =
    (REF FILE f,
     STRING idf,
     CHANNEL ch,
     UNION(STRUCT(INT p, l, c), VOID) mood,
     STRING channelname,
     REF STRING filename) INT:
IF
```



```
      INT x =
         ( mood
         | (STRUCT(INTp, l, c) sz):
              establish(f, idf, ch, p OFsz, l OFsz, c OFsz)
         | open(f, idf, ch)
         ),
      STRINGaction =
         ( mood
         | (VOID): "open"
         | "establish"
         );
      x = 0
   THEN write((newline, "File ", action, "ed with idf """, idf,
            """ on channel """, channel name, """", newline));
      file name := idf;
      0
   ELSE write(( newline,
            action, " failed on output file with idf """,
            idf,
            """ on channel """, channelname, """ returning code ",
            whole(x, 0), newline));
      ( mood
      | (STRUCT(INTp, l, c) sz):
         write((
            "p: ", whole(p OFsz, 0),  ", l: ", whole(l OFsz, 0),
            ", c:", whole(c OFsz, 0), newline))
      );
      x
   FI;

FILE input, output;
STRING in name := "", out name := "";
WHILE
   pr opt := pl opt := ple opt := f opt := lll opt := FALSE;
      g opt := TRUE; s opt := TRUE;
      t opt := c opt := FALSE;
   IF char number(stand in) /= 1 THEN read(newline) FI;
   IF STRING s; rfn(s)
   THEN
      IF s = "" THEN s := "input" FI;
      IF s = in name
      THEN SKIP
      ELSE
         IF in name = "" OR in name = "input"
         THEN SKIP
         ELSE
            close(input);
            write(("Input file with idf """, inname, """ closed.",
               newline))
         FI;
         IF s = "input" THEN input := stand in; in name := ""
```

```
         ELSE
         find file(input, s, standin channel,
            EMPTY,
            "standin channel", in name)
         FI
      FI;
      IF rfn(s)
      THEN
         IF s = out name THEN SKIP
         ELSE
            IF out name = "" THEN SKIP
            ELSE
               close(output);
               write(("Output file closed with idf """, out name, """",
                  newline))
            FI;
            find file(output, s, standback channel,
               STRUCT(INTp, l, c) (1, 1, 10000),
               "standback channel",
               outname)
         FI;
         TRUE
      ELSE print((newline, "No output file name"));
         FALSE
      FI
   ELSE print((newline, "No input file name"));
      FALSE
   FI
)O print((newline, "Input from """, in name, """, ",
         "output to """, out name, """", newline, newline, newline));
   generate parser(input, output,
      pr opt, pl opt, ple opt, f opt, s opt, t opt, c opt, lll opt)
)D;

[F out name /= ""
[HEN close(output);
   print(("Output file """, out name, """ closed. ", newline))
?I;

:ime("Processing ended");
end line

:ND

:ND
```

- - . - -