stichting
mathematisch
centrum

MC

DEPARTMENT OF COMPUTER SCIENCE IW 53/75 DECEMBER

D. GRUNE

THE 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), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

AMS(MOS) subject classification scheme (1970): 68A30

ACM-Computing Reviews-categories: 4.6, 4.12

The MC ALGOL 68 Test Set

by

D. Grune (ed.)

ABSTRACT

The MC ALGOL 68 Test Set comprises 151 ALGOL 68 programs covering the full language with the exception of bulk I/O. Each program has been run on at least one ALGOL 68 compiler.

KEY WORDS & PHRASES: *ALGOL 68, compiler, debugging, parallel processing, Control Data.*

0. Important Note

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 (preferably in the order indicated):

- Tanenbaum, A.S., A Tutorial on ALGOL 68, Computing Surveys, 1976 (to appear).
- Learner, A., Powell, A.J., An Introduction to ALGOL 68 through Problems, Macmillan Computer Science Text, Macmillan, London, 1974.
- Lindsey, C.H., ALGOL 68 with Fewer Tears, Computer J., 15, 176-188, 1972.
- van der Meulen, S.G., Kühling, P., Programmieren in ALGOL 68, Walter de Gruyter, Berlin, 1974 (in German).
- Lindsey, C.H., van der Meulen, S. G., Informal introduction to ALGOL 68 Revised, North Holland Publ. Comp., Amsterdam, 1976 (in preparation).
- Peck, J.E.L., An ALGOL 68 Companion, Univ. of British Columbia, 1972.
- van Wijngaarden, A., et al., editors, Revised Report on the Algorithmic Language ALGOL 68, Acta Informatica, 5, 1-236, 1975.

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 [1] 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 [2], to fulfill 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.

A project like this never comes to an end, so that the moment of publication is arbitrary. I decided to publish the present version since, although it is incomplete, it somehow represents a local maximum of partial completeness: I estimate that the next local maximum is half a man-year away, i.e., about one year in real time. Moreover, with many ALGOL 68 implementations

under way, I think it is more important to publish now than to wait for perfection that can be approached but never be reached.

2. Completeness

The test set is not complete, in two senses. First there is of course the sense 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 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 object 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 for that since such a compiler would often decide not to do any optimization at all. Here a test set is needed based on knowledge of the optimization techniques.

The second sense in which it is not complete is that some aspects of the language are under-represented (e.g. scope testing) or are not represented at all (bulk I/O). These defects could have been remedied but only at the expense of serious postponement of publication.

3. Contents

The test set contains 151 programs, in part correct ones, in part intentionally incorrect ones, distributed over the following categories:

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

- declarers (4 programs)
  tests declarers and their well-formedness.

- mode equivalence (6 programs)

- operators (15 programs)
  tests operator-declarations and calls, including the upb and lwb operators.

- identification (9 programs)
  tests identification of identifiers, mode-indications and operators, including lwb and upb.

- clauses (7 programs)
  if-clauses, case-clauses, displays, vacuums.

- coercions (14 programs)

- identity relations (2 programs)

- stowed values (5 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 (5 programs)
  scope of generators, routine texts and formats.

- jumps (5 programs)

- parallel processing (6 programs)

- simple I/O (9 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 (6 programs)
  various odds and ends that would not fit in other categories.

- application programs (23 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, including numerical work.

4. Readability

Since most readers will probably read large parts of the test set before feeding it to an ALGOL 68 compiler (which may be but should not be lacking), 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
- application 4, 9, 10, 12, 16, 17, 22 and 23.

5. Implementation-independence

Some 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 spite of this I am aware of some dependencies: "establish" is used where "open" would be more appropriate (although other systems might also have quirks in that region), operators not printable in Control Data Display Code (like ~) are lacking almost completely (except in "standard environment 1"), and sometimes a heap-generator may be used where a local-generator would suffice.

Then there are the implementation-dependencies I am not aware of: since this set has been effectively run on only one system they must be present.

6. Reliability

All programs have been processed by the Control Data ALGOL 68 compiler. For most programs this means that they have been tested. A small number of programs, however, could not be tested satisfactorily this way, since they were incompatible with the present version of the Control Data ALGOL 68 compiler.

If the incompatibility was small (either a compiler error or a small documented deviation from the Report), I tested a slightly modified version of the program that circumvented the problem.

Most incompatibilities were due to the absence of restrictions in the compiler that are present in the Report:
No scope checking is done (nor necessary) for references: all generating (including the implicit generating in variable-declarations) is done on the heap.
No transiency testing is done (nor necessary): an object will continue to exist as long as there is still a reference to it or to one of its sub-objects.
No ghost-element is kept (nor necessary): flexibility is carried inwards to all inner modes.
Operator-declarations do not hide operator-declarations in outer blocks: operator-identification identifies the first operator that fits.

In all these cases results (always error messages) were assessed and checked by H.J. Boom, L.G.L.T. Meertens

and me.

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.

Some programs contain timing measurements; these are 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).

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 8, 9 and 10
  these contain all non-bold operators available on the Control Data compiler in Display Code. Other implementations would probably have to make different versions.

- 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 non-output file ("ti") which is then read and checked by the program. It might be useful to have an independent look at this file afterwards.

- standard environment 1
many operators from the standard environment cannot be represented in Control Data Display Code, e.g., window-symbol. For a solution see the preface to the program.

- 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 21
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.

Many useful hints on testing compilers can be found in a paper by R.S. Scowen [3].

8. Availability

The test set can be obtained from the Mathematical Centre in computer-readable form on 7 or 9 track magnetic tape, in ASCII, EBCDIC or Control Data Display Code, as requested. If necessary, paper-tape and Braille versions can be supplied.

9. 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.

Fortunately this test set is not the work of one man; my own contribution is already too large to my taste (about 50 percent) since such large contributions in one style are detrimental to the variation indispensable in a test set.

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, Th.J. Dekker, K. Gostelow, D. Grune, P.G. Hibbard, E. de Jong, J. Kok, L.G.L.T. Meertens, T. Toutenhoofd, F. Teer, 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.

10. Literature

- [1] L. Ammeraal, An Interpreter for Simple ALGOL 68 Programs, IW 7/73, Mathematical Centre, Amsterdam, 1973.

- [2] ALGOL 68 Version I Reference Manual, Control Data Services B.V., Rijswijk, Netherlands, 1975.

- [3] R.S. Scowen, The Diagnostic Facilities in ALGOL Compilers, NPL Report NAC52, July 1974, Teddington, England, 1974.

11. The Test Set

```
  # simple tests  1 #
begin   #algol68 test to see if the compiler exists#
          print("t1");
          int i,j,k;
          int s=17; int t:=3;
          i:= 0;
          for L from 0 by 2 to 13 do i+:= L od;
          print((newline, i,"value of i should be 42"))
end
```

- - . - -

```
  # simple tests  2 #
range of variables:
begin   #test that ranges are correct#
          print ("t2");
          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
```

- - . - -

```
  # simple tests  3 #
begin   #referencing and dereferencing#
          print ("t3");
          int i1=1;
          int i2= 2;
          int i3= 3;
                int ii1:= i1;
                int ii2:= i2;
                      ref int iiii:= ii1;
          print((newline, "value should be 1", iiii));
          iiii:= ii2;
          print((newline, "value should be 2", iiii))
end
```

- - . - -

```
  # simple tests  4 #
multiples:
structures:
begin   #to test structures, multiple values, etc.#
          print ("t4");

          #multiple values#
          [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;
          [1 : 100] real p;
          p[1]:= 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]));

          #test the @ workings#
          p[2:6 #implicit @1#]:= (2.0, 3.0, 4.0, 5.0, 6.0);
          print ((newline, "values are 2.0, 2.0 - 6.0",
                  newline, p[1:6]));
          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]));
          print((newline, "values are 11, 4",
                         upb p[1:3@9], upb p[1:0@5]));
          [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, L));

          #structures#
          struct ([1:2] int m,
                     [1:i[5] # whose value is 5 from above#] real g,
                    bool t) s1, s2;
          t of s1:= t of s2:= L[1,1] = L[1:1,1:2][1,1];  #true#
          for m to upb m of s1 do
          (m of s1) [m]:= (( m of s2) [3-m]:= 50) + 1 od;
          g of s1:= (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, s1,
            newline, "values are 50, 50, 1.0 to 5.0, ''true'':",
            newline, s2));

        #ref of structs#
        [1:2] ref struct ([] int m, [] real g, bool t)
          ssl := (s1, s2);
        print((newline, "values same as before, two times:",
                newline, ssl[1], newline, ssl[2]));
        t of ssl[2]:= false;
        print ((newline, "values are ''true'', ''false'': ",
                t of s1, t of s2))
end
```

- - . - -

```
  # simple tests  5 #
  # simple jumps #
( int j:= 0, i;
  k:i:= j;
    if i >= 2 then goto L fi;
    print("0");
  m:if i >= 1 then        n fi;
    print("0");
  o:goto p;
  L: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 # )
```

- - . - -

```
  # simple tests  6 #
bits bytes strings and other noise:
begin   #don't just stand there, do something!#
```

```
print (("the following are some of the environment enquiries"
        ,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, null character,
    abs null 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 "Ø" fi;
bits c:= bin (abs a + abs b);
print ((s, newline, "answer is",abs c, newline, newline));
if  2r111111 = bin  63
     then skip
     else print ("error in ''bin'' things") fi;

# reduced bytes test #
#bytes are fixed-length strings#
bytes sl:= bytes pack("ab");
[1: bytes width]char cs; #to contain what the
                                        answer should be#
cs[1:byteswidth]:= sl; 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(sl)))
    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 Ø to max abs char
do
    if i mod 64 = Ø
```

```
            then print((newline,
                        whole(i, -4), "-", whole(i+63, -4), " "))
            fi;
            print(repr i)
        od

end
```

- - . - -

```
  # simple tests  7 #
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);

    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
```

- - . - -

```
  # simple tests  8 #
  # 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, L) # no assignation #; L: print ia("+1+2+3       ");
```

```
  ia:= (5, skip, 7); ia[2]:= 6; print ia("+5+6+7        ")
)
```

- - . - -

```
  # simple tests  9 #
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
```

- - . - -

```
  # simple tests 10 #
( int i:=1;
   proc a=(int j) void : print(i+j);
   (int i:=2; a(10).
   )      #  11  #
)
```

- - . - -

```
  # simple tests 11 #
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; L)od; false
        exit L: 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
```

- - . - -

```
  # simple tests 12 #
# 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, ($L"k = "dL, n((2**k+15)over16)(2(2(4(3(d)x)x)x)L)$, k));

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

- - . - -

```
  # simple tests 13 #
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
```

- - . - -

```
  # simple tests 14 #
( # 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 # skip)
```

```
)
```

- - . - -

```
  # simple tests 15 #
  # 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 #
)
```

- - . - -

```
  # simple tests 16 #
#   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, " .");

  print(("test 16  ", newline));
  read(newline);
  read((ch, i, r, b, z, 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"))
)
```

- - . - -

```
  # declarers  1 #
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
```

- - . - -

```
  # declarers  2 #
```

```
begin # shielding, yin & 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
```

- - . - -

```
  # declarers 3 #
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
            Ø
        fi    :    1] int;

    loc hanoi
end
```

- - . - -

```
  # declarers 4 #
(mode u = union(int, real);
      loc union(u) # no list needed # u := 1;
      print(u)
)
```

- - . - -

```
  # mode equivalence  1 #
begin # mode equivalencing #
  mode n = union(struct(real re, im), compl);
  # error, modes are the same #
  skip
end
```

- - . - -

```
  # mode equivalence  2 #
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
```

- - . - -

```
  # mode equivalence  3 #
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
```

- - . - -

```
  # mode equivalence  4 #
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
```

- - . - -

```
  # mode equivalence  5 #
begin # mode equivalencing #
  mode n = union(bytes, bits, ref bits);
                # error, related #
   mode sz = union(int,real,ref union(int,real))
   #  Szeredi - ambiguity  #;
   skip
end
```

- - . - -

```
  # mode equivalence  6 #
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
```

- - . - -

```
  # operators  1 #
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.Ø)  # yields 1 #
end
```

- - . - -

```
  # operators  2 #
begin # operator #
  op + =(union(int, bool)a)int: (a|(bool):1, (int):2);
  print(+if true then true else Ø fi); # 1 #
  print(+if false then true else Ø fi)  # 2 #
end
```

- - . - -

```
  # operators  3 #
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
```

- - . - -

```
  # operators  4 #
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
```

- - . - -

```
  # operators  5 #
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
```

- - . - -

```
  # operators  6 #
begin # operator #
```

```
  op +=(real a, b)real: a-b,
  op +=(ref real a,b)real: a-b; # error, related modes #
  skip
end
```

- - . - -

```
  # operators  7 #
( # 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
)
```

- - . - -

```
  # operators  8 #
( # 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;

  # 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 #
)
```

- - . - -

```
  # operators  9 #
begin # some monadic operators #

  op   &= (int a)int : -a;
  op   += (int a)int : -a;
  op   _= (int a)int : -a;
  op   ?= (int a)int : -a;
  op   ^= (int a)int : -a;
  op  &<= (int a)int : -a;
  op  &>= (int a)int : -a;
  op  &/= (int a)int : -a;
  op  &== (int a)int : -a;
  op  &*= (int a)int : -a;
  op  +<= (int a)int : -a;
  op  +>= (int a)int : -a;
  op  +/= (int a)int : -a;
  op  +== (int a)int : -a;
  op  +*= (int a)int : -a;
  op  -<= (int a)int : -a;
  op  ->= (int a)int : -a;
  op  -/= (int a)int : -a;
  op  -== (int a)int : -a;
  op  -*= (int a)int : -a;
  op  _<= (int a)int : -a;
  op  _>= (int a)int : -a;
  op  _/= (int a)int : -a;
  op  _== (int a)int : -a;
  op  _*= (int a)int : -a;
  op  ?<= (int a)int : -a;
  op  ?>= (int a)int : -a;
  op  ?/= (int a)int : -a;
  op  ?== (int a)int : -a;
  op  ?*= (int a)int : -a;
  op  ^<= (int a)int : -a;
  op  ^>= (int a)int : -a;
  op  ^/= (int a)int : -a;
  op  ^== (int a)int : -a;
  op  ^*= (int a)int : -a;
  op &:== (int a)int : -a;
  op +:== (int a)int : -a;
  op -:== (int a)int : -a;
  op _:== (int a)int : -a;
  op ?:== (int a)int : -a;
  op ^:== (int a)int : -a;
  op&<:== (int a)int : -a;
```

```
op&>:== (int a)int : -a;
op&/:== (int a)int : -a;
op&=:== (int a)int : -a;
op&*:== (int a)int : -a;
op+<:== (int a)int : -a;
op+>:== (int a)int : -a;
op+/:== (int a)int : -a;
op+=:== (int a)int : -a;
op+*:== (int a)int : -a;
op-<:== (int a)int : -a;
op->:== (int a)int : -a;
op-/:== (int a)int : -a;
op-=:== (int a)int : -a;
op-*:== (int a)int : -a;
op <:== (int a)int : -a;
op >:== (int a)int : -a;
op /:== (int a)int : -a;
op =:== (int a)int : -a;
op *:== (int a)int : -a;
op?<:== (int a)int : -a;
op?>:== (int a)int : -a;
op?/:== (int a)int : -a;
op?=:== (int a)int : -a;
op?*:== (int a)int : -a;
op^<:== (int a)int : -a;
op^>:== (int a)int : -a;
op^/:== (int a)int : -a;
op^=:== (int a)int : -a;
op^*:== (int a)int : -a;
op &=:= (int a)int : -a;
op +=:= (int a)int : -a;
op -=:= (int a)int : -a;
op  =:= (int a)int : -a;
op ?=:= (int a)int : -a;
op ^=:= (int a)int : -a;
op&<=:= (int a)int : -a;
op&>=:= (int a)int : -a;
op&/=:= (int a)int : -a;
op&==:= (int a)int : -a;
op&*=:= (int a)int : -a;
op+<=:= (int a)int : -a;
op+>=:= (int a)int : -a;
op+/=:= (int a)int : -a;
op+==:= (int a)int : -a;
op+*=:= (int a)int : -a;
```

```
  op-<=:= (int a)int : -a;
  op->=:= (int a)int : -a;
  op-/=:= (int a)int : -a;
  op-==:= (int a)int : -a;
  op-*=:= (int a)int : -a;
  op_<=:= (int a)int : -a;
  op_>=:= (int a)int : -a;
  op_/=:= (int a)int : -a;
  op_==:= (int a)int : -a;
  op_*=:= (int a)int : -a;
  op?<=:= (int a)int : -a;
  op?>=:= (int a)int : -a;
  op?/=:= (int a)int : -a;
  op?==:= (int a)int : -a;
  op?*=:= (int a)int : -a;
  op^<=:= (int a)int : -a;
  op^>=:= (int a)int : -a;
  op^/=:= (int a)int : -a;
  op^==:= (int a)int : -a;
  op^*=:= (int a)int : -a;

 print(&+-_?^&<&>&/&=&*+<+>+/+=+*-<->-/-=-*_<_>_/_=_*?<?>?/?=?*^<^>^/^=
 ^*&:=+:=-:=_:=?:=^:=&<:=&>:=&/:=&=:=&*:=+<:=+>:=+/:=+=:=+*:=-<:=->:=
 -/:=-=:=-*:=_<:=_>:=_/:=_=:=_*:=?<:=?>:=?/:=?=:=?*:=^<:=^>:=^/:=^=:=
 ^*:=&=:+=:-=:_=:?=:^=:&<=:&>=:&/=:&==:&*=:+<=:+>=:+/=:+==:+*=:-<=:->=:
 -/=:-==:-*=:_<=:_>=:_/=:_==:_*=:?<=:?>=:?/=:?==:?*=:^<=:^>=:^/=:^==:
  ^*=: 1)
  # a 1 should be printed #

end
```

- - . - -

```
  # operators 10 #
begin # some dyadic operators #

  op     &= (inta, b) int : a + b;
  op     -= (inta, b) int : a + b;
  op     _= (inta, b) int : a + b;
  op     ?= (inta, b) int : a + b;
  op     ^= (inta, b) int : a + b;
  op    &<= (inta, b) int : a + b;
  op    &>= (inta, b) int : a + b;
  op    &/= (inta, b) int : a + b;
```

```
op   &== (inta, b) int : a + b;
op   &*= (inta, b) int : a + b;
op   +<= (inta, b) int : a + b;
op   +>= (inta, b) int : a + b;
op   +/= (inta, b) int : a + b;
op   +== (inta, b) int : a + b;
op   +*= (inta, b) int : a + b;
op   -<= (inta, b) int : a + b;
op   ->= (inta, b) int : a + b;
op   -/= (inta, b) int : a + b;
op   -== (inta, b) int : a + b;
op   -*= (inta, b) int : a + b;
op   _<= (inta, b) int : a + b;
op   _>= (inta, b) int : a + b;
op   _/= (inta, b) int : a + b;
op   _== (inta, b) int : a + b;
op   _*= (inta, b) int : a + b;
op   ?<= (inta, b) int : a + b;
op   ?>= (inta, b) int : a + b;
op   ?/= (inta, b) int : a + b;
op   ?== (inta, b) int : a + b;
op   ?*= (inta, b) int : a + b;
op   ^<= (inta, b) int : a + b;
op   ^>= (inta, b) int : a + b;
op   ^/= (inta, b) int : a + b;
op   ^== (inta, b) int : a + b;
op   ^*= (inta, b) int : a + b;
op  &:== (inta, b) int : a + b;
op  +:== (inta, b) int : a + b;
op  -:== (inta, b) int : a + b;
op  _:== (inta, b) int : a + b;
op  ?:== (inta, b) int : a + b;
op  ^:== (inta, b) int : a + b;
op &<:== (inta, b) int : a + b;
op &>:== (inta, b) int : a + b;
op &/:== (inta, b) int : a + b;
op &=:== (inta, b) int : a + b;
op &*:== (inta, b) int : a + b;
op +<:== (inta, b) int : a + b;
op +>:== (inta, b) int : a + b;
op +/:== (inta, b) int : a + b;
op +=:== (inta, b) int : a + b;
op +*:== (inta, b) int : a + b;
op -<:== (inta, b) int : a + b;
op ->:== (inta, b) int : a + b;
```

```
op -/:== (inta, b) int : a + b;
op -=:== (inta, b) int : a + b;
op -*:== (inta, b) int : a + b;
op _<:== (inta, b) int : a + b;
op _>:== (inta, b) int : a + b;
op _/:== (inta, b) int : a + b;
op _=:== (inta, b) int : a + b;
op _*:== (inta, b) int : a + b;
op ?<:== (inta, b) int : a + b;
op ?>:== (inta, b) int : a + b;
op ?/:== (inta, b) int : a + b;
op ?=:== (inta, b) int : a + b;
op ?*:== (inta, b) int : a + b;
op ^<:== (inta, b) int : a + b;
op ^>:== (inta, b) int : a + b;
op ^/:== (inta, b) int : a + b;
op ^=:== (inta, b) int : a + b;
op ^*:== (inta, b) int : a + b;
op  &=:= (inta, b) int : a + b;
op  +=:= (inta, b) int : a + b;
op  -=:= (inta, b) int : a + b;
op  _=:= (inta, b) int : a + b;
op  ?=:= (inta, b) int : a + b;
op  ^=:= (inta, b) int : a + b;
op &<=:= (inta, b) int : a + b;
op &>=:= (inta, b) int : a + b;
op &/=:= (inta, b) int : a + b;
op &==:= (inta, b) int : a + b;
op &*=:= (inta, b) int : a + b;
op +<=:= (inta, b) int : a + b;
op +>=:= (inta, b) int : a + b;
op +/=:= (inta, b) int : a + b;
op +==:= (inta, b) int : a + b;
op +*=:= (inta, b) int : a + b;
op -<=:= (inta, b) int : a + b;
op ->=:= (inta, b) int : a + b;
op -/=:= (inta, b) int : a + b;
op -==:= (inta, b) int : a + b;
op -*=:= (inta, b) int : a + b;
op _<=:= (inta, b) int : a + b;
op _>=:= (inta, b) int : a + b;
op _/=:= (inta, b) int : a + b;
op _==:= (inta, b) int : a + b;
op _*=:= (inta, b) int : a + b;
op ?<=:= (inta, b) int : a + b;
```

```
  op  ?>=:= (inta, b) int : a + b;
  op  ?/=:= (inta, b) int : a + b;
  op  ?==:= (inta, b) int : a + b;
  op  ?*=:= (inta, b) int : a + b;
  op  ^<=:= (inta, b) int : a + b;
  op  ^>=:= (inta, b) int : a + b;
  op  ^/=:= (inta, b) int : a + b;
  op  ^==:= (inta, b) int : a + b;
  op  ^*=:= (inta, b) int : a + b;

  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, -/=:=1, -==:=1,
   -*=:=1, _<=:=1, _>=:=1, _/=:=1, _==:=1, _*=:=1, ?<=:=1, ?>=:=1,
   ?/=:=1, ?==:=1, ?*=:=1, ^<=:=1, ^>=:=1, ^/=:=1, ^==:=1, ^*=:=1;

print(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->=:1-/=:1-==:1-*=:1_<=:1_>=:1_/=:1_==:1_*=:1?<=:1?>=:
  1?/=:1?==:1?*=:1^<=:1^>=:1^/=:1^==:1^*=:1)
  # this formula should deliver 109 #
end
```

- - . - -

```
  # operators 11 #
begin # some operators, dyadic only #

  op    <= (inta,b)int:a+b;
  op    >= (inta,b)int:a+b;
  op    /= (inta,b)int:a+b;
  op    == (inta,b)int:a+b;
  op    *= (inta,b)int:a+b;
  op   .<<= (inta,b)int:a+b;
```

```
op   <>= (int a,b) int: a+b;
op   </= (int a,b) int: a+b;
op   <== (int a,b) int: a+b;
op   <*= (int a,b) int: a+b;
op   ><= (int a,b) int: a+b;
op   >>= (int a,b) int: a+b;
op   >/= (int a,b) int: a+b;
op   >== (int a,b) int: a+b;
op   >*= (int a,b) int: a+b;
op   /<= (int a,b) int: a+b;
op   />= (int a,b) int: a+b;
op   //= (int a,b) int: a+b;
op   /== (int a,b) int: a+b;
op   /*= (int a,b) int: a+b;
op   =<= (int a,b) int: a+b;
op   =>= (int a,b) int: a+b;
op   =/= (int a,b) int: a+b;
op   === (int a,b) int: a+b;
op   =*= (int a,b) int: a+b;
op   *<= (int a,b) int: a+b;
op   *>= (int a,b) int: a+b;
op   */= (int a,b) int: a+b;
op   *== (int a,b) int: a+b;
op   **= (int a,b) int: a+b;
op  <:== (int a,b) int: a+b;
op  >:== (int a,b) int: a+b;
op  /:== (int a,b) int: a+b;
op  =:== (int a,b) int: a+b;
op  *:== (int a,b) int: a+b;
op <<:== (int a,b) int: a+b;
op <>:== (int a,b) int: a+b;
op </:== (int a,b) int: a+b;
op <=:== (int a,b) int: a+b;
op <*:== (int a,b) int: a+b;
op ><:== (int a,b) int: a+b;
op >>:== (int a,b) int: a+b;
op >/:== (int a,b) int: a+b;
op >=:== (int a,b) int: a+b;
op >*:== (int a,b) int: a+b;
op /<:== (int a,b) int: a+b;
op />:== (int a,b) int: a+b;
op //:== (int a,b) int: a+b;
op /=:== (int a,b) int: a+b;
op /*:== (int a,b) int: a+b;
op =<:== (int a,b) int: a+b;
```

```
op =>:== (inta,b)int:a+b;
op =/:== (inta,b)int:a+b;
op ==:== (inta,b)int:a+b;
op =*:== (inta,b)int:a+b;
op *<:== (inta,b)int:a+b;
op *>:== (inta,b)int:a+b;
op */:== (inta,b)int:a+b;
op *=:== (inta,b)int:a+b;
op **:== (inta,b)int:a+b;
op  <=:= (inta,b)int:a+b;
op  >=:= (inta,b)int:a+b;
op  /=:= (inta,b)int:a+b;
op  ==:= (inta,b)int:a+b;
op  *=:= (inta,b)int:a+b;
op <<=:= (inta,b)int:a+b;
op <>=:= (inta,b)int:a+b;
op </=:= (inta,b)int:a+b;
op <==:= (inta,b)int:a+b;
op <*=:= (inta,b)int:a+b;
op ><=:= (inta,b)int:a+b;
op >>=:= (inta,b)int:a+b;
op >/=:= (inta,b)int:a+b;
op >==:= (inta,b)int:a+b;
op >*=:= (inta,b)int:a+b;
op /<=:= (inta,b)int:a+b;
op />=:= (inta,b)int:a+b;
op //=:= (inta,b)int:a+b;
op /==:= (inta,b)int:a+b;
op /*=:= (inta,b)int:a+b;
op =<=:= (inta,b)int:a+b;
op =>=:= (inta,b)int:a+b;
op =/=:= (inta,b)int:a+b;
op ===:= (inta,b)int:a+b;
op =*=:= (inta,b)int:a+b;
op *<=:= (inta,b)int:a+b;
op *>=:= (inta,b)int:a+b;
op */=:= (inta,b)int:a+b;
op *==:= (inta,b)int:a+b;
op **=:= (inta,b)int:a+b;

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(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**=:1)

  # the number 91 should be printed #
end
```

- - . - -

```
  # operators 12 #
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
```

- - . - -

```
  # operators 13 #
begin # operator test, illegal #

  op +=<== (int a) int :-a;                                  # bad #

  op <==(inti) int : -i;                        # <= is dyadic only #
```

```
  op +==(inta,b,c)int:1;
  op +==:=(inta,b,c)int:1;
  op ===(inta)int:1;
  op ==:=(inta)int:1;

  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
```

- - . - -

```
  # operators 14 #
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, 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 boxes 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)
```

```
end
```

- - . - -

```
  # operators 15 #
begin
  # incorrect, since not all boxes are of the mode row-of #
  print(upb union ([] int, bool) ([] int (1)));
  print(lwb union (ref [,] string, string) ("ab"))
end
```

- - . - -

```
  # identification 1 #
L1: if int i:=1; false then int i:= 2; print(i)     else
  print(i) #1# fi
```

- - . - -

```
  # identification 2 #
L2: if int i:=1; true then int i:= 2; print(i) #2# else
  print(i)     fi
```

- - . - -

```
  # identification 3 #
begin int i = 1;

  proc a = void : ( int i = 2; b);
  proc b = void : print(i);

  a # +1 #
end
```

- - . - -

```
  # identification 4 #
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 voide) void :
    ( int i = 4, j = -4; e);

  a # -2 #
end
```

- - . - -

```
  # identification 5 #
begin int i:= 1; (int i = i; print(i) # what is the value of i #)
end
```

- - . - -

```
  # identification 6 #
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.Ø, n, rreal, 1.Ø, n, real, 3.Ø, n, rreal, 3.Ø, n,
    rreal, 2.Ø, n, rreal, 2.Ø, n, int, 3, n, real, 3.Ø, n,
    rreal, 3.Ø, n, real, 4.Ø, n, rreal, 4.Ø, n, real, 4.Ø, n,
    rreal, 4.Ø, 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.Ø, rr out 4.Ø 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
```

- - . - -

```
  # identification 7 #
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 #
  print(lwb (1| (8, 2), 3, []real: skip));  #  6 #

  skip
end
```

- - . - -

```
  # identification 8 #
begin # hiding of operators #

  # 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.Ø)  # ok, ok #; print(newline));

  (op + = (union([]int, []real) i ) int : 3;
                                                  # does not hide #
    print(+ 1); print(+ 1.Ø)  # ok, ok #; print(newline));

  (op + = (union([]int, real) i ) int : 3;        # hides #
    print(+ 1); print(+ 1.Ø)  # ko, ok #; print(newline));

  (op + = (ref union(int, bool) i ) int : 3;     # hides #
    print(+ 1); print(+ 1.Ø)  # ko, ko #; print(newline));

  (op + = (union(char, ref union(int, bool)) i ) int : 3;
                                                      # hides #
    print(+ 1); print(+ 1.Ø)  # ko, ko #; print(newline));

  (op + = (union(char, ref union(ref int, ref bool)) i )
              int : 3;                          # does not hide #
    print(+ 1); print(+ 1.Ø)  # ok, ok #; print(newline));

  skip
end
```

- - . - -

```
  # identification 9 #
( # obscuring lwb and upb #

  (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
)
```

- - . - -

```
  # identification 10 #
begin for i from 1 by i do skip  od
  # second i is unknown #
end
```

- - . - -

```
  # clauses 1 #
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
```

- - . - -

```
  # clauses 2 #
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
```

- - . - -

```
  # clauses 3 #
begin # wrong case clauses #

  union (int, bool, real) ibr = skip;

  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;                                                 #ok#

  case case ibr in (union (int, real) ir) : ir esac
  in (bool) : ibr
  esac;                                                 #ko#

  skip
end
```

- - . - -

```
  # clauses 4 #
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
```

- - . - -

```
  # clauses 5 #
(  #test vacuum as string #

  proc p = (string s) void:
  print((newline, lwb s, upb s, s));

  p("") # +1  +Ø  #;
  p(()) # +1  +Ø  #;
  p(begin  end) #1 Ø#
)
```

- - . - -

```
  # clauses 6 #
begin # vacuum #
  []int i=(); print(i[1]) # runtime error, overflow #
end
```

- - . - -

```
  # clauses 7 #
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
```

- - . - -

```
  # coercions  1 #
begin # coercions #
    print ((real x:= 0; ref [] real (x):= 1; x)); #1#
    print ((int n:= 0; n +:= 1:= 5)) #5#
end
```

- - . - -

```
  # coercions  2 #
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, 3i5 esac,
         im of case n in 1, 2.0, 3i5 esac))
  od
                                        # 1.0 0.0, 2.0 0.0, 3.0 5.0 #
end
```

- - . - -

```
  # coercions  3 #
```

```
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
```

- - . - -

```
  # coercions  4 #
begin # coercion error, a unit is not a coercend #
  [] struct (int i, bool j) k = ((1), (true));
  skip
end
```

- - . - -

```
  # coercions  5 #
begin # row display cannot be united #
  print(upb if false then []int(1) else (1,2,3) fi)
end
```

- - . - -

```
  # coercions  6 #
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
```

- - . - -

```
  # coercions  7 #
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  Ø #
end
```

- - . - -

```
  #  coercions 8 #
begin # soft balance #
    print((real x:= 3.14;
           case 3 in
           skip,
           if x<Ø then stop else
           ref [] real : nil fi,
           if x>Ø then x else x+:=1 fi
           esac:=pi)[1])
    # 3.14159265...#
end
```

- - . - -

```
  # coercions  9 #
begin # soft balance #
    print (case 2 in skip, nil,
                     if bool (skip) then stop
                     else proc ref [] int (skip)
                     fi
           esac :=:
        case 3 in loc ref ref [] int, loc int, nil esac
          )
    #true, would you believe #
end
```

- - . - -

```
  # coercions 1Ø #
begin # union with void #

  op ? = (int i) proc int : int : 1;
  op & = (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:= ? 1;
  pupiv;

  upiv:= ^ 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
```

- - . - -

```
  # coercions 11 #
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 : loc int:= 1), newline));
  print((ref[,]int(ref[]int : loc[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
```

- - . - -

```
  # coercions 12 #
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
```

- - . - -

```
  # coercions 13 #
begin # soft balance with exits #

  int i; [ 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
```

- - . - -

```
  # coercions 14 #
( # rowing of nil yields nil #
    print(("print: ", true, " ",
              ref[]int(nil) :=: ref int(nil), newline)))
```

- - . - -

```
  # identity relations  1 #
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
```

- - . - -

```
  # identity relations  2 #
begin
    if int i, j, k, L; i:=:j and k:=:L # illegal formula #
    then skip fi;

    real a;
    a :=: (L);              # correct, L = ref real #
    a :=: L;                # incorrect, L is a unit, not a tertiary #
    L: skip
end
```

- - . - -

```
  # stowed values 1 #
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
```

- - . - -

```
  # stowed values 2 #
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+:=1od 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][3:15][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
```

- - . - -

```
  # stowed values 3 #
```

```
begin [1:-1] int k; print("ok"); k[1]:= 1 # overflow # end
```

- - . - -

```
  # stowed values 4 #
begin
  print([][] bool(true, 2r1)) ; # tf...ft #
  print(newline);
  print([,] bool(true, 2r1)) # runtime error, wrong length #
end
```

- - . - -

```
  # stowed values 5 #
(     ""[];   # ok #
      "a"[];  # ko #
      "ab"[]  # ok #
)
```

- - . - -

```
  # flexibility/transiency 1 #
( # 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))
)
```

- - . - -

```
  # flexibility/transiency 2 #
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))

end
```

- - . - -

```
  # flexibility/transiency 3 #
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
```

- - . - -

```
  # flexibility/transiency 4 #
(     # ghost element: prints 3, then crashes #

      for k from 3 to 4
      do flex [1:0] [1:3] int flex fix;
           flex fix:= loc [1] [1:k] int # ok only of k=3 #;
           print(k)
      od
)
```

- - . - -

```
  # generators & garbage collection  1 #
begin # test garbage collector #
  to 1000 do heap [ 1000 ] real od;
  print(("collections, garbage, collect seconds:", newline));
  print(( collections, garbage, collect seconds  , newline))
end
```

- - . - -

```
  # generators & garbage collection  2 #
begin # test garbage collector #
  ref [] real x, y, int n:=0;
L: x:= heap[1:1000] real; y:= heap[1:1000] real;
  if (n+:=1)<1000 then L fi;
  print(("collections, garbage, collect seconds:", newline));
  print(( collections, garbage, collect seconds  , newline))
end
```

- - . - -

```
  # generators & garbage collection  3 #
begin # test garbage collector #

  int size = 250;

  ref [] real x, [1: size ] ref [] real y, int n:= 1;
L: 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 L fi;

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

end
```

- - . - -

```
  # generators & garbage collection  4 #
begin # heap #
  int n:= 0, ref int x:= loc int :=0;
 L: (heap int p:= n+:= 1; print((x, p)); # 0,1,1,2,2,... #
  x:= p); (n<100|L);
  print(newline);
  print(("collections, garbage, collect seconds:", newline));
  print(( collections, garbage, collect seconds  , newline))
end
```

- - . - -

```
  # scope  1 #
begin # scope error #

  proc pp = (int i) proc int: int : i + 1 # error #;
  print(pp(1))

end
```

- - . - -

```
  # scope  2 #
begin #scope error#
    proc void pv= (L: void:
        (mode ml= [1:($n((L; heap int):= 3) "a" $; 1)] int;
        ml x:= 1; skip ));
    pv
end
```

- - . - -

```
  # scope  3 #
begin # no scope error #
    proc void pv= (L: void:
        (mode ml= [1:($n(( #L;# heap int):= 3) "a" $; 1)] int;
        ml x:= 1; skip ));
    pv
end
```

- - . - -

```
  #  scope 4 #
  # routine scope error #
  begin


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
```

- - . - -

```
  # scope 5 #
  # 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
)
```

- - . - -

```
  # jumps  1 #
  # simple jumps and exits #
( for i to 2 do
      if i = 2 then goto L fi; print("first") exit
    L:print(" second")
  # result:  first second #od )
```

- - . - -

```
  # jumps  2 #
begin real a; goto L; int i:= 1; L: print(i)
  # the declaration of i has not been elaborated #
end
```

- - . - -

```
  # jumps  3 #

begin # jump #
  int i:=1, j:=2;
```

```
  i:= j:= (L; 3); L: print((i, j)) # 1  2 #
end
```

- - . - -

```
  # jumps  4 #
( # jump out of procedure #

 ( # directly #
  proc jump = void: ( print(2); print((9, 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 #
```

- - . - -

```
  # jumps  5 #
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
    voids, 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; took 65 sec. in ALGOL 60 on the EL-X8 #

    bits bits # proposed pattern # := bin 0,
         acc  # assembled pattern #;

    real time:= clock, 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;
    time:= clock - time;
    if cnt /= 2 ** max width - 1 then
        print(("something wrong", cnt, 2 ** max width - 1)); stop
    fi;
    print(("stack jump test successful, time taken: ", time, newline))
end
```

- - . - -

```
  # parallel processing   1#
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 stack swapping.
#

# 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
```

- - . - -

```
  # parallel processing 2 #
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 L),
        # the sorters # build sorters (n sorters)
    end; L: 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((p,newline));
  sort(p); print((p,newline))
);
```

```
  test((int p) int: p);
  test((int p) int: entier (p/5));
  test((int p) int: 0)
end
```

- - . - -

```
  # parallel processing  3 #
( # simple deadlock # sema s = level 0;
      void par(down s, downs); print("escaped")
)
```

- - . - -

```
  # parallel processing  4 #
( # uninitialized sema, will it wreck the run-time system ? #
      par(down loc sema, down loc sema); print("escaped")
)
```

- - . - -

```
  # parallel processing 5 #
( # action on sema outside par-environment #
  sema s = level 1; down s; print(("escaped once", newline));
  down s; print(("escaped twice", newline))
)
```

- - . - -

```
  # parallel processing 6 #
( # sema with negative initial value #
  sema ten = level -10;
  par((down ten; print(" second")),
    (to 10 do up ten od; print("first"); up ten)
  )
)
```

- - . - -

```
  # simple I/O  1 #
```

```
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", char compress 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 preceeding space#, newline, "*" #no
                  preceeding 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 lets 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", 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: #end of test#
        copy to stand out:
        print((newpage, newline, j,x,c,t,rj,"*", newline,"*", newline,
               "x", newline, "*", backspace, "x", newline,s));
        to upb s do backspace(standout)od;
        print((s2,backspace,s2,newpage))
end
```

- - . - -

```
  # simple I/O  2 #
begin

  print(if true then 3 else 3.Ø fi);              # 3 #
  print(newline);
  print(if false then 3 else 3.Ø fi);             # 3.Ø #
  print(newline);
  print(if true then (3, 3.Ø) else
                   (2r11, true, newline) fi);          # 3 3.Ø #
  print(newline);
  print(if false then (3, 3.Ø) else
                   (2r11, true, newline) fi);          # f...fttt #
  print((int i:= 1; (i + 1, i - 1, newline) # coll. clause #
            # serial clause # ));                            # 2 Ø #

  print(union([] int, bool) ([] int (1)));          # 1 #
  print(newline);
  print(union([] int, bool) (bool (true)));       # true#
  print(newline);
    # print works on a union of everything, so also on bool or []int #

  print(()); print("empty");                                  # empty #
  print(newline);

  print((3, sqrt( goto L ),5));  print("error");
L:print("correct jump out of print parameter")
end
```

- - . - -

```
  # simple I/O 3 #
begin
# 1Ø/Ø8/73, R van Vliet; 3Ø/Ø9/75, 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.Ø,
    (long long real): long long max real -leng leng 2.Ø
  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
```

- - . - -

```
  # simple I/O 4 #
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 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 beginnig of the line# od;
    print((new line,
           "this line should be preceeded 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
```

- - . - -

```
  # simple I/O 5 #
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: out);
        do put(f, space) od;
    out: i:= char number(f) -1; to i do put(f, backspace) od; i);

  print(("test 3", new line,
    "some tests on proc(ref file)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 : out);
                  ff);
        proc nlp =(ref file f) void:
          new line(f);
        nlp(rf);
        int half width=max ch n ? 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,
         "wasn''t it fine?", new line))
    )
end
```

- - . - -

```
  # simple I/O 6 #
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
```

- - . - -

```
  # simple I/O 7 #
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
```

- - . - -

```
  # simple I/O 8 #
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
```

- - . - -

```
  # simple I/O 9 #
( # parameter of "print" #

      print((1, 2.0, 3i4, "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, 3i4, "5", "67", true, 16r89, newline))
);

( # parameters of printf #
  printf(($ 3d L $, 1, union(int, format) (2)))

))
```

- - . - -

```
  # standard 1 #

  # standard operators #
    # Some characters used in chapter 10 do not exist as such in cdc
      display code. They are represented here by two crosses followed by
      one display code character. These combinations can be used for
      editing purposes  for running on other computers.
      It concerns the following characters:
```

```
      or symbol                    ##o
      and symbol                   ##a
      differs from symbol          ##=
      is at most symbol            ##<
      is at least symbol           ##>
      over symbol                  ##:
      percent symbol               ##p
      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 ##p i, newline));
print((i over i, newline));
print((i ##:### i, newline));
print((i ##:* i, newline));
print((i ##p### i, newline));
print((i ##p* 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 ##p:= i, newline));
print((ii modab i, newline));
print((ii ##:##*:= i, newline));
print((ii ##:*:= i, newline));
print((ii ##p##*:= i, newline));
print((ii ##p*:= 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))),
            (print(("first", level pv, newline)); up pv)
           )
  );

      skip
end
```

- - . - -

```
  # standard 2 #
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 false
  ; reidf possible(f)
then skip
  ; 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 false
  ; set possible(f)
then skip
  ; print(((set(f, 1, 1, 1); "set"), newline))
else print(("no set", newline)) fi;
if false
  ; reset possible(f)
then skip
  ; 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

end
```

- - . - -

```
  # standard 3 #
begin co all format items co

      int i = 2; union(int, real) uir = 2;
      print(("10.3.4.1. literals and insertions", newline));
      printf(($ d L $, 1));
      printf(($ # comment # d L $, 1));
      printf(($ co comment co d L $, 1));
      printf(($ comment comment comment d L $, 1));
      printf(($ x 2(d) L $, 1));
      printf(($ x k d L $, 1));
      printf(($ x x d L $, 1));
```

```
printf(($ x y d L $, 1));
printf(($ x p d L $, 1));
printf(($ x q d L $, 1));
printf(($ x "one" d L $, 1));
printf(($ x 2"one" d L $, 1));
printf(($ x "one" 2"two" d L $, 1));
printf(($ x "one""two" d L $, 1));
printf(($ x "one"1"two" 2x "three" d L $, 1));
printf(($ x "one"1"two" 2y "three" d L $, 1));
printf(($ x "aa" n(i)y d L $, 1));
printf(($ x "aa" nbegin i endy d L $, 1));
printf(($ x "aa" n( true | i )y d L $, 1));
printf(($ x "aa" nif true then i fiy d L $, 1));
printf(($ x "aa" n( i | i, i )y d L $, 1));
printf(($ x "aa" ncase i in i, i esacy d L $, 1));
printf(($ x "aa" n( uir | (int): i )y d L $, 1));
printf(($ x "aa" ncase uir in (int): i esacy d L $, 1));
printf(($ x "aa" n(heap int:= i)y d L $, 1));
printf(($ x "show" n(int:jmp n)y "do not show" dl $,1)); jmp n:
print(("end of show", newline));
printf(($ x "a" d L $, 1));
print(newline);

print(("10.3.4.2. integral patterns", newline));
printf(($ x d L $, 1));
printf(($ x sd L $, 1));
printf(($ x z L $, 1));
printf(($ x sz L $, 1));
printf(($ x zdzd L $, 1));
printf(($ x z2dzd L $, 1));
printf(($ x z2sdzd L $, 1));
printf(($ x z2sdsz "a"d L $, 1));
printf(($ x z+sdsz "a"d L $, 1));
print(newline);

print(("10.3.4.3. real patterns", newline));
printf(($ x d . d L $, 1.0));
printf(($ x d s. d L $, 1.0));
printf(($ x d "s"s. d L $, 1.0));
printf(($ x d . L $, 1.0));

printf(($ x d . d e + d L $, 1.0));
printf(($ x d . d se + d L $, 1.0));
printf(($ x d . d "a"se + d L $, 1.0));
printf(($ x d . e + d L $, 1.0));
```

```
printf(($ x d e + d L $, 1.0));
print(newline);

print(("10.3.4.4. boolean patterns", newline));
printf(($ x b L $, true));
print(newline);

print(("10.3.4.5. complex patterns", newline));
printf(($ x d.d i d.d L $, compl(1, 1)));
printf(($ x d.d si d.d L $, compl(1, 1)));
printf(($ x d.d "a"si d.d L $, compl(1, 1)));
print(newline);

print(("10.3.4.6. string patterns", newline));
printf(($ x aa L $, "xx"));
printf(($ x asa L $, "xx"));
printf(($ x 2a L $, "xx"));
printf(($ x a "a"sa L $, "xx"));
print(newline);

print(("10.3.4.7. bits patterns", newline));
printf(($ x 2r d L $, 2r1));
printf(($ x 4r d L $, 2r1));
printf(($ x 8r d L $, 2r1));
printf(($ x 16r d L $, 2r1));
printf(($ x 2r sd L $, 2r1));
printf(($ x "a"2r d L $, 2r1));
print(newline);

print(("10.3.4.8. choice patterns", newline));
printf(($ x c ( "a", co c co 2"a"1"p", "bcd") L $, 2));
printf(($ x "z"c ( "a", co c co 2"a"1"p", "bcd") L $, 2));

printf(($ x b ( "a", co c co 2"a"1"p") L $, false));
printf(($ x "z"b ( "a", co c co 2"a"1"p") L $, false));
print(newline);

print(("10.3.4.9. format patterns", newline));
printf(($ x f ($ d L $) $, 1));
printf(($ x f if true then $ d L $ fi $, 1));
printf(($ x f case 1 in $ d L $, skip esac $, 1));
printf(($ x f case uir in (int):$ d L $ esac $, 1));
print(newline);

print(("10.3.4.10. general patterns", newline));
```

```
      printf(($ x g L $, 1));
      printf(($ x "z"g L $, 1));
      printf(($ x g(2) L $, 1));
      printf(($ x g(4,1) L $, 1));
      printf(($ x g(7,1,2) L $, 1));

      printf(($ x g L $, 1.Ø));
      printf(($ x "z"g L $, 1.Ø));
      printf(($ x g(2) L $, 1.Ø));
      printf(($ x g(4,1) L $, 1.Ø));
      printf(($ x g(7,1,2) L $, 1.Ø));

      printf(($ x g(heap int:=7,heap int:=1,heap int:=2)L$,1));
      printf(($ x "show" g(int:jmp g) "do not show" L $, 1)); jmp g:
      print(("end of show", newline));

      skip
end
```

- - . - -

```
  # syntax errors 1 #
begin # 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);

  (op is = (int i) bool : i = Ø; is 1);
  (op of = (int i) bool : i = Ø; of 1);
  (op at = (int i) bool : i = Ø; 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);
  (long # ko # 2r1Ø1);

  # go to is allowed, but watch the loop-clause #
  (go to exit; go # ko # to exit; go
    to exit);
  (for i from go to exit do skip od);
  (from go to exit do skip od);

  skip
end
```

- - . - -

```
  # syntax errors 2 #
begin [] real x # this is a small program, the rest is missing#
```

- - . - -

```
  # syntax errors 3 #
    :
```

- - . - -

```
  # syntax errors 4 #
end
```

- - . - -

- - . - -

```
  # syntax errors 6 #
```

```
  # comment without end
```

- - . - -

```
  # syntax errors 7 #
  # please feel free to shuffle #
-one);print((valueofa,valueofbs//t,dp*dq*r)end;,m;(k<L;r//:=t;(p
/-q)|begin intr=nof<Ø|-i/-j|:j=(ratp,q)<=nofqcase formu=
leftoperandofet,v=rightoperandofet;formvalueof=(forme)real:
caseein(ref constec):valueofec,(ref varev):valueofev,(
ref op-:==(ref ratp,ratq)ref rat:p:=p-q,op*:==(ref ratp
,rat constc;valueofc:=valueofec-1;c)*udash)esac,(ref callef
):ri,=Ø|error|intr=i&j;q)ref rat:p:=p*q,op/:==(ref ratp,rat
q)ref rat:p:=p/q;op=p&nofq,s,u/v,exp(v*ln(u))esac,(ref call
ef):begin ref functionf=functionnameofef;valueofboundvaroff=(
ratp,q)bool:nintj)ref int:ioverab bool:nofp*dofq;(s//t,
dp*dq*r)end;op-=(ratp,q)rat:begin intnp=nofp,nq=nofq&d
ofp;((nofp//r)*(nofq//s),(dofp//s:=1;valueofx:=1;f:=a+x/(b+x)
;gudash=derivativeof(u,x),vdash=derivativeof(v,x);operatorofetin
udash+vdash,udash-vdash,u*vdash*dofp;op*=(ratp,q)rat:begin
intr=nofp&dofq,s=nofq:=dofq;intr:=dp&=(ratr)real:real
(nofr)/real(dofr)b:=("b",skip),x:=("x",skip);valueofa:=1;
valueofb|m:=L;L:=k;k:=m);n:m:=kmodL;(m=Ø|L|k:=L;L;op&=(inti,j)
int:abs(i=Ø|j|:j=ofp=nofqanddofp=dofq;op/==(ratp,q)
bool:nofp/=nofqordofp,valueofx,valueof(derivativeof(g,x))))
end begin mode rat=struct(intn,d);prio//=7,p,nq=nofq;int
dp:=dofp,dq:=dofq;intr:=dp&dq;dp;intdp:=dofp,dqbegin ref
functionf=call:=(fdash,g))*derivativeof(g,x)end esac;proc:=
valueof(parameterofef);valuep//s)*(nof;op+=(ratp,q)rat:
begin intnp=nofØ|i|:i=lorj=1|1|intk:=i,L:=j):v*u**(heap ref
vary=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=Ø|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<Ø|-(=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=Ø|j|:j=Ø|i|:i=lorj=1|1|intk:=i,L:=j,m;(k<L|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,rat)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<==(:=L;L:=k;k:=m);n:m:=kmodL;(m=Ø|L|k:=L;L:=m;n));op/=(inti,
j)rat:(j<Ø|-i/-j|:j=Ø|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=Ø|error|:q<Ø|-(
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;((
```

- - . - -

```
  # syntax errors 7 #
  # 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 # ' #,
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)
    than 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 pdv d_ od;
    p
);
proc letter= (char ch) bool: "a" <= ch and ch <= "z";
```

```
proc digit= (char ch) bool: "Ø" <= 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);
(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:= Ø
);
þ‡ø¢ 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);
‡¢f text no text = nil;
ref text first text:= no text, last text:= no text;
on logical file end (progam, (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 od 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:= Ø; last text:= first text;
    while last text :/vd no text
    do inf cnt:= Ø, ref text p:= last text;
        to range (2 * mean -1)
        do (p:/=: no text
            y pdv next oof p; cnt +:=1)
        od # determine chunk #;
        # enter into chunk chain #
```

```
        last chunk:=
        (last chunk:=: no chunk
        y first chunk
        | next of last chunk):=
        heap chunkcdv 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;
        yo 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
    newline(resultef close (result);
    printf(($"produced" 4 zdx, "chunks of mean length"
        3zdL$, n chunks, mean));
    open (result, "result", z type channel)
 end
end
```

- - . - -

```
  # miscellaneous misery  1 #
begin

  # comments #
  int i:= 1; #huppeldepupco i:= 2; co puppup # print(i) #1# ;

  #denotation, test precision#
  print(3.14159265358979323846264338327) #pi# ;
  print(newline);

  #denotations #
  print(
  ( 0, 1, 01, 00000000000000000002, newline,
```

```
   01.02, .0102, 01.02\0, .0102\0, 01.02e0, .0102e0, newline,
   .01020\+2, .0102\-0, 0102\-2, 0102e-04,
   01.02e-0000000000000000000000000, .0102e+00, newline,
   "t", "h", "e", " above two lines ", "should be id"
   "entical", """did that work?"""))
end
```

- - . - -

```
  # miscellaneous misery  2 #
( # format denotation ? #
  format f =
    (# cp # 1, # count # 0, # bp # 1, # c # () )
    # or something else likely to fool your compiler # ;
  putf(stand out, f)
)
```

- - . - -

```
  # miscellaneous misery  3 #
(   # ALGOL 68 programma
      (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)
)
```

- - . - -

```
  # miscellaneous misery  4 #
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)[1][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
```

- - . - -

```
  # miscellaneous misery 5 #
# runs and yields 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
```

- - . - -

```
  # miscellaneous misery  6 #
begin # Test recursivity 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 Function
    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
end
```

- - . - -

```
  # application  1 #
begin # ALGOL 68 program, Th.J.Dekker 730705
    calculates all increasing sequences
    adding up to a given integer from 1 to 10 #

    [1 : 4] int a;

    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;
    proc print solution = (int p) void:
    print( (a[1:p], newline) );

    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
```

- - . - -

```
  # application  2 #
begin # ALGOL 68 program, Th.J.Dekker 730706
    calculates all increasing sequences adding up
    to a given integer from 1 to 10.
    For programs yielding the same output see
    ALGOL 68 program Th.J.Dekker 730705
    and Th.J.Dekker, Syllabus Informatica, ITW, UVA, 1972, p.81-82 #

    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
        [] intst = 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
```

- - . - -

```
  # application  3 #
begin # ALGOL 68 program Th.J.Dekker 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
```

- - . - -

```
  # application  4 #
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 # ;

##

    proc test = (real x0, y0, proc(real)real f, string s,
        union (string, real) sol)
    void:
    print((newline, newline, s, newline, x0, y0,
        if real x, y;
            zero in(x:= x0, y:= y0, f,
                (real p) real :1  \-10 + 1\-10 * abs p)
        then (x, f(x)) else "no solution found" fi,
        "       on EL-X8: ", sol)) #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
```

- - . - -

```
  # application  5 #
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 #

  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,
            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; anew fi;
```

```
    x0:= x4; f0:= f4
  end #of int#;

  int; sum / 180
end #of qad hr#;

proc function = (real x, k) real:
  (eval +:= 1; x <= 0 | 0 | exp(ln(x) * k));
int eval;

print(("results for fully recursive version:", newline, newline));
print("         time         ");
print("        exponent       integral                error        ");
print("             skip           points ");
print(newline);
 for k from 4 to 10
 do int skip; eval:= 0;
   real time:= clock;
   real z = qad fr(0, 1, (real x) real: function(x, k),
                   (1e-8, 1e-9, skip));
   time:= clock - time; print(time);
   print(( k, z, z - 1 / (k + 1), skip, eval, newline))
 od ;

print(newline);
 for k from 2 to 7
 do int skip; eval:= 0;
   real time:= clock;
   real z = qad fr(0, 1, (real x) real: function(x, 1 / k),
                   (1e-8, 1e-9, skip));
   time:= clock - time; print(time);
   print(( k, z, z - k / (k + 1), skip, eval, newline))
 od;


print((newline, newline));
print(("results for half-recursive version:", newline, newline));
print("         time         ");
print("        exponent       integral                error        ");
print("             skip           points ");
print(newline);
 for k from 4 to 10
 do int skip; eval:= 0;
   real time:= clock;
   real z = qad hr(0, 1, (real x) real: function(x, k),
```

```
                        (1e-8, 1e-9, skip));
      time:= clock - time; print(time);
      print(( k, z, z - 1 / (k + 1), skip, eval, newline))
    od ;

  print(newline);
   for k from 2 to 7
   do int skip; eval:= 0;
     real time:= clock;
     real z = qad hr(0, 1, (real x) real: function(x, 1 / k),
                        (1e-8, 1e-9, skip));
     time:= clock - time; print(time);
     print(( k, z, z - k / (k + 1), skip, eval, newline))
   od

  #the answers should approximately be:
   group 1: .200000,.166667,.142857,.125000,.111111,.100000,.090909
   group 2: .666667,.750000,.800000,.833333,.857143,.875000
  #
end
```

- - . - -

```
  # application  6 #
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:= ∅, minmn:= (m < n | m | n), pk:= 1,
    real w, eps, sigma:= ∅, 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 < ∅ | w | - w); a[k,k]:= akk - aidk;
       beta:= - 1 / (sigma - akk * aidk); pk:= k; sigma:= ∅;
      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
```

- - . - -

```
  # application  7 #
begin # numerical 1 #
```

```
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
```

- - . - -

```
  # application  8 #
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 kl = k + 1;
            p[k]:= pk; r:= k;
            if pk /= k
            then ichvec(a[ :k-1,k], a[ :k-1,pk]);
                 ichvec(a[k,kl:pk - 1], a[kl: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:= kl;
        for i from kl 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 #;

real tijd:= clock;
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, le-13) = n
     then solsym(a, piv, b);
        for i to n do print(real (aa[i,] * b));
           print(real(2 ** (n - i))); print(newline)
        od
     else print("coefficients matrix is not positive definite")
     fi;
     print(newline); print(" time used : ");
     print(real(clock - tijd)); print(newline)
```

```
     od
    #output approximately: 1
                           2  1
                           4  2  1
                           . . .
                           256   128   . . .   1 #
end
```

- - . - -

```
  # application 9 #
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
```

- - . - -

```
  # application 10 #
begin # collateral sorting #

  proc quicksort = (ref [] item a) void:
    # quicksort requires the operator < to be defined for two items #
  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)
end
```

- - . - -

```
  # application 11 #
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 # example of formula manipulation #
```

- - . - -

```
  # application 12 #
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 entryl:= nil;
      for i to upb entry
      do ref ref tag entry i = entry[i];
        if entry i :/=: ref tag (nil) then
          if ( entryl :=: ref ref tag (nil) | true |
              tag of entry i < tag of entryl) then
            entryl:= entry i
          fi
        fi
      od;
      entryl :/=: ref ref tag (nil)
    do act( tag of entryl, info of entryl);
      ref ref tag(entryl):= chain of entryl
    od
  );

comment
proc 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;
print(("end debug;", newline))
);
comment

  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)))
end
```

- - . - -

```
  # application 13 #
rat: # Dik Winter, 141075#
begin #handling of rationals#
    mode rat = struct (int n, d);

    #preliminary routines#
    op // = (int i, j) int: i over j;
    prio // = 7;
    op & = (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;
        LL: k:= ii - ii // jj * jj; ii:= jj; jj:= k;
            if jj = 0 then ii else LL fi
        fi;
    op / = (int i, j) rat:
        begin int k = i & 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 & 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 & k, d = dp * dq;
        (n // L, k // L * d)
    end;
op - = (rat q, p) rat:
    begin int k = d of q & 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 & 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 & dp, j = np & 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 & np, j = dp & dq;
        if np >= ∅
        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 & 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 & 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 & 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 & 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 & dp, j = np & 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 & np, j = dp & dq;
        q:= if np >= ∅
            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 & i;
        (i // k * n of q, dq // k)
    end;
op / = (rat q, int i) rat:
    begin int nq = n of q; int k = nq & i;
        if i >= ∅
        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 & 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 & i;
        q:= if i >= ∅
            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 & i;
        (i // k * n of q, dq // k)
    end;
op / = (int i, rat q) rat:
    begin int nq = n of q; int k = nq & i;
        if  nq >= ∅
        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 < ∅ | "-(" |: width > ∅ | "+(" | "(") +
        whole(abs n of q, ∅) + "/" + whole(d of q, ∅) + ")";
        width = ∅
    then s
    else if int us = upb s, aw = abs width;
            us > aw
        then aw * (q < ∅ | "-" | "+")
        else (aw - us) * " " + s
        fi
    fi;

#innerproduct of two arrays of rationals#
op +* = (ref [] rat a, b) rat:
    begin rat s:= (∅, 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:= (∅,1), int kl:= 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:kl] +* aik[1:kl];
                if piv = ∅ and aik[i] /= ∅
                then piv:= aik[i]; pk:= i
                fi
            od;
            if piv = ∅
            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:kl] +* a[1:kl,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
end
```

- - . - -

```
  # application  14   #
begin  # 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; [ max ] int p;

  proc  test= (proc (int) int a) void:
  ( for i to max do p[i]:= a(i)od;
    shuffle(p); print((p,newline));
    sort(p); print((p,newline))
  );

  test((int p) int: p);
  test((int p) int: entier (p/5));
  test((int p) int: 0)
end
```

- - . - -

```
  # application  15   #
begin

   proc dig char=(int x)char:
   "0123456789abcdef"[x+1];
   print(dig char(7));                                    # 7 #
   print(digchar(12));                                    # c #

   proc char in string=(char c,ref int i,string s)bool:
   (bool found:=false;
   for k from lwb s to upb s while not found
   do (c=s[k]|i:=k; found:=true) od;
   found);

   int i;
   char in string("c",i,"abcd");print(i);                 # 3 #
```

```
proc char dig=(char x) int:
(x=" "|0|
  int i;
  char in string(x,i,"0123456789abcdef"); i-1);
print((char dig("7"), char dig("0"), char dig("f"), newline));
                                                    # 7 0 15 #

char errorchar="*";

mode number=union(int,real);

proc subwhole=(number v,int width) string:
case v in (int x):
    (string s, int n:=x;
    while dig char(n mod 10) +=: s; n overab 10;
       n ne 0
     do skip od;
     (upb s > width | width*errorchar | s))
esac;
  print(("/", subwhole(123, 8), "/", newline));   #/123/#
  print(("/", subwhole(123, 2), "/", newline));   #/**/ #

proc whole=(number v,int width) string:
case v in
    (int x):
 (int length:=
   abs width-(x<0 or width >0 |1|0),
  int n:=abs x;
 if width = 0 then
  int m:=n; length:=0;
  while m overab 10; length+:=1; m ne 0
    do skip od
 fi;
  string s:=subwhole(n,length);
 if length=0 or char in string(errorchar,loc int,s)
 then abs width*errorchar
 else
   (x<0|"-"|:width>0|"+"|"")+=: s;
  (width ne 0|(abswidth-upb s)*" " +=:s);
  s
 fi),
 (real x):"......."
esac;
print((newline,"whole",newline));
for width from 5 by -1 to -5
```

```
   do
      print((width,"    /",whole(+123,width),"/",newline))
   od
end
```

- - . - -

```
  # applications 16 #
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 400               # minus 400-year correction #
           + []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 + 5) 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))

end
```

- - . - -

```
  # application  17   #
comment this is the intended program
begin # ALGOL 68 program Th.J.Dekker 730702.
         this program tests some operator calculus and print the same
         results as the ALGOL 68 program Th.J.Dekker 730701, 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

end
comment # end of intended program #

begin # 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
```

- - . - -

```
  # application 18 #
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 ?:= 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 ?:= 2 od;
         sequence);

      for c from 0 to 100
      do print((c, length(c), sequence(c), newline,
              code(sequence(c)), newline, newline))
      od
end
```

- - . - -

```
  # application  19   #
          #   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
```

- - . - -

```
  # application  20   #

begin co JKok, 70110,
   queens on chessboard in ALGOL68 co

   int aantal, real tijd := clock,

   proc output = (ref [] int dame ) void :
   begin int n = upb dame;
      print(newline); print(" ");
      for i to 4 * n - 1 do print("-") od;
      for i to n
      do int k = dame[i]; print(newline);
         for j to n
         do if j = k then print("| q ") else print("|   ") fi
         od;
         print("|");
         print(newline); print(" ");
         for i to 4 * n - 1 do print("-") od
      od;
      to 2 do print(newline) od; aantal +:= 1
   end co print board of size n * n co;

   for n from 3 to 8
   do [1 : n] int dame,

      proc zet = (int LL, kk) void :
      if int L:= LL + 1, k; dame[LL]:= kk; LL = n
      then output(dame)
      else for i from ( dame[1] = 1 | 2 |:
           (dame[1] < L) and (n - dame[1] > L - 2) | 1 | 2)
           to ( (dame[1] <= L) and (n - dame[1] > L - 1)
            | n | n - 1 )
           do bool test:= false;
              for j to L - 1 while not test
```

```
            do test:= (k:= dame[j]; i = k or L - j = i - k
               or L - j = k - i)
            od;
            if not test then zet(L, i) fi
         od
   fi co move queen L to square k co;

   aantal:= 0;
   for i from 1 to n over 2 do zet(1, i) od;
   print(("number of solutions for n=", n, " is", aantal));
   print(newline); print("time used : ");
   print(real(clock - tijd)); print(" sec."); print(newline);
   tijd:= clock
od
end
```

- - . - -

```
  # application 21 #

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 the
file "input". 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

  file program; # contains the program #
  establish(program, "program", z type channel, 1, 10000, 80);
  file result; # will contain the minced program #
  establish(result, "result", z type channel, 1, 10000, 80);

  int line width = 72; # for program and result #
  string result sep = "////////////////////////////"; # for result #
  char quote = """", bold = repr 39 # ' #;

  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" or s = bold + "comment" + bold;

  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 = bold
```

```
  then int p = last(letgit);
    (line[c pos: p] + bold,
       p +
        (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], 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";

proc digit = (char ch) bool: "Ø" <= 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 file 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", "") L $,
      n chunks, mean, sep))

  od
end
```

- - . - -

```
  # application 22 #
begin # sheep in mountain cleft #
```

```
  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
```

- - . - -

```
  # application 23 #
begin # All-parser, Dick Grune, 20-11-74.
          The following is an examle 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, real time:= clock;
    t(act: count+:=1);
    time:= clock - time;
    print((count, time, newline))
  od
end
```

- - . - -