Raabo.

A/S REGNECENTRALEN

SCANDINAVIAN INFORMATION PROCESSING SYSTEMS

RCSL No: 53-S7
Edition: October 1970
Author: Søren Henckel

Title: recordinput
Part 1

Keywords: RCSL Statistical Package, Input of data in Records, Input of control Information, Syntax Check of Input.

Abstract: The boolean procedure recordinput reads one input paper tape containing a number of records. The records are syntactically checked and will be delivered on a backing storage for later inspection. Only syntactical errors are detected, whereas semantics has to be checked in a later scan of the output file. 17 pages.

DK-2500 VALBY · BJERREGAARDSVEJ 5 · TELEPHONE: (01) 46 08 88 · TELEX: 64 64 rcinf dk · CABLES: INFOCENTRALEN

CONTENTS:

APPENDIX:

1. FUNCTION AND PURPOSE.

The procedure recordinput attempts to solve the problems concerning input of control information to the RCSL statistical package and has been made in order to fulfil the following general rules:

1: It should be possible to divide any input paper tape into a number of logical records.

2: You may delete any record containing anything.

3: In every possible error situation it should be possible to give a correct picture (list) of the erroneous record.

4: All not deleted records has to appear in the output file generated by the procedure, whereas all erroneous or deleted records has to be copied on current output mixed with error messages.

5: The procedure should only check for syntax errors, whereas semantic errors has to be detected in a later scan.

6: The entire data paper tape has to be read by calling the procedure once.

Procedure heading:

```
external boolean procedure
recordinput(maxchar,maxparam,realtext,realname,descriptor,tailcontent);
value       maxchar,maxparam,realtext,realname;
integer     maxchar,maxparam,realtext,realname;
real array                                    descriptor;
integer array                                           tailcontent;
```

Call parameters:

integer maxchar = maximum number of non blind characters in one input record
(maxchar must be >= 5)

integer maxparam = maximum number of parameters in an output record
(maxparam must be >= 1)

integer realtext = maximum number of reals used for storing a text parameter

integer realname = maximum number of reals used for storing a name parameter
(realtext and realname must be >= 0)

real array descriptor(1:number_of_descriptors)
each array element must at call contain one descriptor as a short string (e.g. either the entire descriptor or, for long descriptors, exactly the 5 first characters and a null character).

integer array tailcontent(-1:number_of_descriptors)
the array elements numbered 1 until number_of_descriptors must at call contain an integer number code (1 <= code <= 5) showing what the tail of the corresponding kind of record is intended to contain. Explanation of the code is:

code = 1 <=> a number of text parameters (at least one)
code = 2 <=> exactly one non empty name parameter
code = 3 <=> a number of number parameters (mixed reals and integers)
code = 4 <=> empty (e.g. only separators)
code = 5 <=> end of input tape (a possible tail is ignored).

integer array elements
tailcontent(-1) = number of (different) descriptors
tailcontent(0) = number of segments reserved on backing storage for output of records (must be >= 0)

Return parameters:

boolean recordinput true if no troubles with the output file has occured, otherwise false.

If the procedure return value is false, none of the call parameters have been touched, otherwise we have the following return parameters:

real array descriptor(1:2) name (generated by the monitor) on the backing storage file used for output of records.

integer array elements

tailcontent(0) = number of accepted records in the <input file>
tailcontent(1) = total number of records read by the procedure

The rest of the call parameters are unaltered.

Check of call parameters:

The procedure checks that the call parameters are reasonable but a complete check is not performed.

2.1. INPUT/OUTPUT FILES.

In order to facilitate the use of procedure recordinput, the standard zones in/out (connected to current input/output) has been selected as input file for data on character level and output file (on character level) for error messages, whereas the records accepted or generated by the procedure are placed on a backing storage file with a name generated by the monitor (and delivered as return parameter).

The format of error messages is A4 vertical with approx. 60 lines on each page with each line containing maximally 71 characters starting in position 1.

Because the usually used printers only have one kind of letters and the File Processor requires small letters and in order to minimize the number of error situations, all letters in the <input file> are transformed to small letters.

The notion <record> may need further explanation: In order to have input partitoned in logical parts and to facilitate the preparation of control information, it is comfortable to let the kind of information be given if front of the information, and among many possibilities, a verbal description has been prefered. This means that every record begins with a text showing what the rest of the record is about. An example may be:

ident: output identification<

where ident shows, that the rest of the record (to the character <) has to be used as an identification text in output.

2.2. INPUT SYNTAX.

<input file> ::= <record>$^{*}_{1}$

<record> ::= <accept record>|<delete record>|<EM record>

<accept record> ::= <descriptor><parameter>$^{*}_{0}$<accept>

<delete record> ::= <any string without <terminator> ><delete>

<EM record> ::= <any string without <terminator> ><EM>

<terminator> ::= <accept>|<delete>|<EM>

<accept> ::= <separator>$^{*}_{0}$ < <between records>

<delete> ::= <separator>$^{*}_{0}$ > <between records>

<EM> ::= <end of medium character>

<between records> ::= <<separator>|<|>>$^{*}_{0}$

<separator> ::= ,|;|:|SP|NL|HT|VT|FF|<separator><separator>

<descriptor> ::= <name>

<parameter> ::= <number parameter>|<text parameter>|<name parameter>

<number parameter> ::= <separator><number>

<number> ::= <leading part>$^{1}_{0}$<decimal part>$^{1}_{0}$<exponential part>$^{1}_{0}$

<leading part> ::= <sign>$^{1}_{0}$<digit>$^{*}_{0}$

<decimal part> ::= <decimal point><digit>$^{*}_{1}$

<exponential part> ::= <exponent mark><sign>$^{1}_{0}$<digit>$^{*}_{1}$

<name parameter> ::= <separator><name>

<name> ::= <letter><<letter>|<digit>>$^{*}_{0}$

<text parameter> ::= <text start><text>

<text start> ::= : <separator without <:>>$^{*}_{0}$

<text> ::= <legal character without <separator> <:> and <terminator>>
<legal character without <:> and <terminator>>$^{*}_{0}$|empty

<legal character> ::= <character in the normal ISO set which is available on printer without <question mark>>

Further the part of an <accept record> which is not the <descriptor> is in the following denoted as the tail.

2.3. FORMAT OF OUTPUT FILE

The output file will contain a number of records consisting of real elements in the basic format:

If the descriptor is of kind number k and the number of parameters in the record is p, the format of the output record is:

element index	content
1	p+2 as real (converted by assignment)
2	k as real (- - -)
3	parameter(1)
4	parameter(2)
	.
	.
	.
p+2	parameter(p).

The procedure adds three new kinds of descriptors to the kinds given in the call:

1: kind of descriptor = number of descriptors + 1
is used for indicating errors in records where the determination of the kind of descriptor has been impossible. These records have parameter number:= 0;

2: kind of descriptor = number of descriptors + 2
is used for indicating errors in the tail of a record with known kind of descriptor.
These records have parameter number := 1; and
parameter(1):= found kind of descriptor;

3: kind of descriptor = number of descriptors + 3
indicates an end of data record (code in tailcontent = 5) or an <EM record>. This record (exactely one from each successful call of the procedure) has parameter number := 0;

The format of output records containing a number of texts (code in tailcontent array = 1) needs a special explanation:

These records contain 2 real elements showing (as usual) the total recordlength and the kind of record. Further they contain a number of subrecords (exactely one subrecord for each textparameter in the input record).

If the textparameter occupies l reals, the length of the subrecord is l + 1 real elements. In the first of these elements the textlength (=l) is placed as a real, and the rest of the subrecord contains the textparameter packed with 6 characters in each real element (and a null character placed after the last textcharacter).

The records are outrec'ed in such a way, that every zone buffer contains an entire number of records.

2.4. FORMAT AND EXPLANATION OF ERROR MESSAGES.

If a record is not deleted (by >), the procedure first tries to determine the kind of descriptor (e.g. kind of record). If this succeeds, the syntactical check is continued according to general rules and the number code in the corresponding element of the call parameter tailcontent.

If an error is detected (or if the record has been deleted) the procedure will produce an error message supplied with a character picture of the erroneous (or deleted) record.

The error message format is:

record number<<-d> <error message>
copy of record:<character list of record>

In the <character list of record> the total record will be printed with these modifications:

(a) the characters NL, FF, HT and VT will in list be replaced by SP.

(b) the list is made with 71 maximally characters on each line.

The records in <input file> are counted 1,2,3 and so on, and the error message output is partitioned in pages numbered 1,2,3 and so on (each page will contain approx. 60 lines = format A4 vertical, with output starting in printing position number 1).

The 30 different error messages are made in such a way, that I mean they need no further explanation. An example of an error message can be:

28 10 70 record input syntax errors in data page 3

record number 34 has some syntax error in a number parameter
copy of record: model 73 75-<

3. PROGRAM TAPE, STORAGE REQUIREMENTS AND RUNNING TIME.

The procedure recordinput is written in ALGOL 5 as an external boolean procedure (see ref. 1), and is available as algol text on 8 channel paper tape both in the normal ISO form (with even parity) and in flexowriter mode (with odd parity); these tapes must be loaded by <i tre> and <i trf> respectively.

All letters in the program text are small letters because the normally used printers only have one kind of letters.

The structure on RC 4000 does not permit giving exactely times for compilation and running time, but some general rules can be mentioned:

The compilated external procedure occupies 19 segments on backing storage, and reading and compilation takes approx. 25 seconds.

Compilation and running may be done in a normal 10 k bytes process, but it will be much more comfortable to run the program in a 15 k bytes process. With maxparam < 126 and maxchar < 1000 it is possible to run the program in a 10 k bytes process whereas maxparam = 250 and maxchar = 2500 requires approx. 15 k bytes. These sizes are minimal, and it is possible to cut down running time with approx. 50 pct. by using respectively 15 k and 20 k bytes.

Normally the control information is not very large and therefore the speed of the procedure is not very critical, but as a general rule you may say, that the procedure reads 1 page of control information in about 5-15 seconds (real time) dependent on the number of erroneous records (which has to be copied in the error list).

4.1. METHOD.

The general rule about character list of erroneous records in any situation (even in case of illegal numbers) combined with the claim on the possibility of deletion at any time and the claim on unique partitioning of any input tape in logical records are the reasons why input is performed on character level by means of the integer procedure class_of_input. This procedure reads one character from zone in and places it (if possible) in the boolean array character(1:maxchar+1) for later inspection. Further this procedure checks the basic syntax (illegal characters, EM character and long record).

If the record is an <accept record> the kind of descriptor is determined according to the following rule: if the character string expected to be the <descriptor> contains more than 5 characters, only the first 5 characters are used in textmatching for determining the kind of descriptor, otherwise the total descriptor character string is used (e.g. the three strings <variables>, <variavzig> and <variab> are all equivalent with the given descriptor string <varia>).

If the record is terminated by <delete> the procedure gives a deletion message and a character picture of the deleted record.

If the record is not deleted and the determination of the kind of descriptor has succeded, the conversion of the tail is performed by scanning the characters stored in the boolean array character(tailstart:position).

The number parameters are converted in four parts: <leading part>, <decimal part>, <exponential part>, and <sign> which later are combined by the statement

number:= (leading part + decimal part) × exponential part × sign;

This way of conversion causes the following conditions to be true

0 <= abs(leading part) <= 8388599,

0 <= decimal part × 10 ↑↑ number of decimals <= 8388599 and

0 <= abs(exponent) <= 599.

Numbers with syntax as +'-3 are converted correctly whereas 0.0'1, 0.0'1 and 0'1 all gives wrong conversion to 10. All other ALGOL 60 numbers are correctly converted.

The name- and text parameters are packed with 6 characters in each real element. After the last character in each parameter (possibly in the next real element) is placed a null character in order to facilitate the later use of these parameters.

4.2. PROGRAM STRUCTURE.

The procedure has been programmed in such a way that alterations are as easy as possible to make. All variables (except i, j and text) have names which shows their use.

The two main procedures are

1: integer procedure class_of_input

2: integer procedure unpack_character.

which are used for character input and conversion respectively.

The procedure error (error_type) is large and rather slow because of the 30 long error texts combined with the character list of the erroneous record, and the decision of the further error reactions such as creating parameters for possible error records.

The entire procedure body of procedure recordinput consists of one block containing several local blocks each one containing declarations of very local quantities.

After initializing and finding (with check) the date of run, the call parameters are checked in order to avoid errors not covered by the control in running system (without index check). Next a backing storage file, with name generated by the monitor (and delivered as return parameter) and size as given in the call parameter tailcontent(-1), is created (if possible).

Initializing of the alphabet stored in the integer array table(0:127) is done according to the following scheme:

input class	contents
0	DEL, NULL
1	empty
2	digits (0, 1, ... , 9)
3	sign (+ -)
4	decimal point (.)
5	exponent mark (')
6	small and capital letters (a,.., å, A,.., Å)
7	empty
8	graphics(= / exclamation mark 'and')(× _ and '')
9	illegal characters (the rest) shown as <question mark>
10	terminators (< > and EM)
11	normal separators (; , SP NL FF HT and VT)
12	text start (:)

The active part of the procedure is from label action: in line 220 to label finish_input: in line 585. This part is formed as a case statement controlled by the integer variable state. Each case refer to a state within reading or conversion of one record.

Case 1 and 2 is reading (and packing) descriptor and tail respectively.
Case 3, 4, and 5 corresponds to accept-, delete-, and end- (of data) record respectively,
Case 6 is conversion of the tail and contains 5 subcases corresponding to the 5 possible different number codes in the call parameter tailcontent. The third subcase (conversion of numbers) is a case statement containing 8 cases (controlled by the integer variable

state, which here means state within one number parameter).
Case 7 is outrec of the syntactically correct and converted record. The records are made in such a way that each zone buffer contains an entire number of complete records.
Case 8 is used when a record contains more than maxchar non blind characters. The list is continued until a <terminator> is met and the further action is decided.
Case 9 is used in case of error in call parameters, problems with create entry or by call of the block procedure error_in_doc.

The total action scheme is:

state	action	new state
1	read descriptor	2,3,4,5,8,9
2	read tail	3,4,5,8,9
3	determine descriptor in accept record	6,7
4	list delete record	1
5	end (of data) action	9, -
6	convert the tail	7,5
7	outrec of (correct) record	1, 9
8	list long record	4,5,7
9	unspecified problems	-

5. REFERENCES.

(1) Søren Lauesen: ALGOL 5. User's manual
A/S Regnecentralen Copenhagen. July 1969.

(2) Søren Lauesen: File Processor. User's manual
A/S Regnecentralen Copenhagen. April 1968

(3) Per Brinch Hansen: Multiprogramming System
A/S Regnecentralen Copenhagen. April 1968.

6.1. USER'S EXAMPLE, PROGRAM.

The following program has been used for control of procedure recordinput:

```
test program for recordinput  of  28 10 70

begin
  zone z(256,2,stderror);  integer array t(-1:9);
  real array d(1:12);  integer i,j,k,m;
  message  søh test record input of 28 10 70;
  for i:= 9 step -1 until 1 do
  begin
    t(i):= case i of (5,3,2,3,3,1,3,1,4);
    d(i):= real ( case i of ( <:end:>,<:list:>,<:data:>,
    <:outpu:>,<:model:>,<:comme:>,<:varia:>,<:head:>,<:execu:>))
  end;
  t(-1):=15;  t(0):=9;  d(10):=real <:unknown kind:>;
  d(11):= real <:error in known kind:>;  d(12):=d(1);
  if recordinput(250,35,12,2,d,t) then
  begin
    i:= j:= 1;  open(z,4,string d(increase(i)),0);
    write(out,<:<12>            list from area:      :>,
    string d(increase(j)));  m:= 0;  d(2):= real <:list:>;
new:
    i:= inrec(z,2);  j:= z(1)-2;  if j<0 then
    begin
      write(out,
      <:<10><10>          segment change.  rest=:>,<<-d>,i+2);
      inrec(z,i);  goto new
    end;
    i:= z(2);  inrec(z,j);  m:= m+1;
    write(out,<:<10><10>record number:>,<<-d>,m,
    <:  of kind number:>,i,<: = :>,string d(i),<:<10>:>);

    case case i of (0,3,4,3,3,2,3,2,1,1,5,6) of
    begin
      write(out,<:empty record. length =:>,<<-d>,j);
      begin
        write(out,<:text record. length =:>,<<-d>,j);  k:= 1;
        for k:= k+1 while k<=j do
        write(out,<:<10>length =:>,<<dd>,z(k-1),<:::>,
        string z(increase(k)),<:::>)
      end;
      begin
        write(out,<:number record. length =:>,<<-d>,j);
        for i:= 1 step 1 until j do
        write(out,if i mod 6=1 then <:<10>:> else <::>,
        << -dddd.dd10-dd>,z(i))
      end;
      begin
        i:= 1;  write(out,<:name record. length =:>,<<-d>,j,<:::>,
        string z(increase(i)),<:::>)
      end;
```

```
      write(out,<:found kind =:>,<<-d>,z(1),<: = :>,string d(z(1)));
      begin
        write(out,<:end of control file<12>:>);  goto finish
      end
    end case;
    goto new;
finish:
  end if record input;
  close(z,true)
end program
```

6.2. USER'S EXAMPLE, INPUT TAPE.

The test run has been performed using this <input file>, punched on 8 channel paper tape in flexowriter mode:

```
Test data for recordinput of 28 10 70>
data testfile70<
head  :  1970 experiments (regression of height/age):
;text number 2<
Variables  7,  4a<ouTput 1 -7 ₁₀-4,,,1.036₁₀12<
variab  ,7,,  4,,,;<
head:,a:;B:
C::E:<
execution<
erroneous kind of descriptor (erroneous is illegal)<
10. record shows deletion of hard errors>
execu-tion<
Head experiment number 2<     variables  86043780<
head :  080446/0519 = number code for output identification <
     variables<
execu:<
varia 3,7.   4<   data:version3<variables 3 7.0 4<
end of testdata for recordinput containing 20 records<
```

6.3. USER'S EXAMPLE, OUTPUT.

From this we obtain the following of output:

```
17 11 70      record input          syntax errors in data          page 1


record number 1 has been deleted in input (by  >)
copy of record: test data for recordinput of 28 10 70>


record number 4 has some syntax error in a number parameter
copy of record: variables  7,  4a<
```

```
record number 9 has an illegal kind of descriptor
copy of record: erroneous kind of descriptor (erroneous is illegal)<


record number 10 has been deleted in input (by  >)
copy of record: 10. record shows deletion of hard errors>


record number 11 contains an illegal character in the descriptor
copy of record: execu-tion<


record number 12 has no text start (:) before the first text
copy of record:  head experiment number 2<


record number 13 has overflow in leading part of a number parameter
copy of record:      variables  86043780<


record number 17 has empty digit part in the decimal part of a number
copy of record: varia 3,7.   4<


record number 20 is an end of input record.   input finished.


survey from record_input:

total number of records in input was 20
and of these were 12 accepted.


          list from area:      wrk000015

record number 1  of kind number 3 = data
name record. length = 2:testfile70:

record number 2  of kind number 8 = head
text record. length = 13
length = 8:1970 experiments (regression of height/age):
length = 3:text number 2:

record number 3  of kind number 11 = error in known kind
found kind = 7 = varia

record number 4  of kind number 4 = outpu
number record. length = 4
  1000.00₁₀ -3 -7000.00₁₀ -3  1000.00₁₀ -7  1036.00₁₀  9
```

```
record number 5  of kind number 7 = varia
number record. length = 2
  7000.00₁₀ -3  4000.00₁₀ -3

record number 6  of kind number 8 = head
text record. length = 12
length = 1:a:
length = 1:b:
length = 1:c:
length = 1::
length = 1:e:
length = 1::

record number 7  of kind number 9 = execu
empty record. length = 0

record number 8  of kind number 10 = unknown kind
empty record. length = 0

record number 9  of kind number 10 = unknown kind
empty record. length = 0

record number 10  of kind number 11 = error in known kind
found kind = 8 = head

record number 11  of kind number 11 = error in known kind
found kind = 7 = varia

record number 12  of kind number 8 = head
text record. length = 12
length = 9:080446/0519 = number code for output identification  :

record number 13  of kind number 7 = varia
number record. length = 0

record number 14  of kind number 9 = execu
empty record. length = 0

record number 15  of kind number 11 = error in known kind
found kind = 7 = varia

record number 16  of kind number 3 = data
name record. length = 2:version:

record number 17  of kind number 7 = varia
number record. length = 3
  3000.00₁₀ -3  7000.00₁₀ -3  4000.00₁₀ -3

record number 18  of kind number 12 = end
end of control file
```