Program Generators for Logical Checking of Statistics and Statistical Table Construction

(A paper indited by a group of research workers from the Central Statistical Board—Romania)

The Central Statistical Board is working out a high level language which will facilitate the specification of the logical checking requirements and of the statistical table structure in a very simple way. The object of this paper is to explain the characteristics of this language.

1. INTRODUCTION

Statistical processing programming using electronic computers can be very well automatized, since, in most cases, two operations have to be performed: (1) the preparation of record files and (2) their use in drawing up tables. To this end, two systems of program generators are worked out: The first generates logical checking programs which will create correct archives in order that the final tables should not include incompatibilities. The second system contains a table program generator and a set of auxiliary programs which perform the additional processing and the table editing.

Taking into account the general characteristics of such processing and in order to simplify the language, a number of restrictions have been imposed; however, these restrictions have no special practical implications and can be avoided.

Besides the programs included into the system, a set of forms will be prepared; they are necessary to define and analyze the problem.

The languages and forms agree with each other in a way which will permit to pass easily from the analysis to the elaboration of source programs.

*

In view of elaborating these programs, the following definitions are needed:

Individual: information unit, containing a set of characteristics, formally considered a collection of indicators.

Population: a set of persons (individuals).

Indicator: a characteristic i of an individual, e.g.: i: $P \rightarrow V_i$ where P is population and V_i is a set called value indicators.

Semantically, the indicators can be classified as follows:

qualitative indicators — expressing the individual's affiliation to a certain class, or serving to identify the individual within a population;

quantitative indicators — expressing a certain quantitative property of an individual.

Qualitative indicators of identification are used to identify the individual.

From the representation point of view the indicators are:

(a) numerical;

(b) alphanumerical.

Table: Numerical matrix.

Cell: The element of a matrix.

Affiliation criterion of the cell: Formula defining individuals belonging to a certain cell.

Cell tabulation criterion: Formula defining the quantity (or the indicator value) to be added in the cell whenever occurs an individual satisfying the cell affiliation criterion.

Row (column) affiliation criterion: Formula defining individuals belonging to a certain row (column).

Note. A cell may have only one tabulation criterion. Usually all the cells in a row (column) have the same tabulation criterion. Therefore, the tabulation in a table criterion must be specified either for rows or for columns.

Archive: A record file corresponding to the investigated population. Every record file corresponds to an indicator.

2. THE LOGICAL CONTROL GENERATOR

Purpose of the program

The logical control generator is a program generating logical control programs as source programs, compilable with # XPLJ program. The language accepted by the generator contains all the PLAN 3 language instructions and other special instructions in "COLOG", the latter being a specific language for the logical control.

Structure of the generated program

The generated program will input a file on MT and punched cards; the file data are then logically controlled. The output file records will have a unique structure, compatible with the tabulation programs.

The generated program will accept a second input for the rejected cards; the output file will receive another name, of an unchanged structure. The rejected records are also printed.

A third entry of the program will allow the introduction of some constants from cards into a standard area of the store. The structure of a program is shown below.

PROGRAM NAME/PRIORITY	
PERIPHERALS	*
DIRECT AREAS	
DIRECT AREAS	
INDIRECT AREAS	*
INPUT FILE DESCRIPTION MT	*
INPUT FILE DESCRIPTION PC	*
OUTPUT FILE DESCRIPTION MT	
OUTPUT FILE DESCRIPTION LP	*
PROGRAM 0/1	
PROGRAM 2	
PROGRAM 3/7	

The program functions

Entry 0 reads records from MT if the input file is on MT, if not, reads from cards. For each record the PROGRAM 0/1 is run and, depending on the accuracy of the data, the record is kept on the output tape, or is rejected and listed by the line printer.

The input file contains records of decimal characters (as the cards are).

When the information of a census unit does not fit one card, it may be punched on two or more cards.

While this, the information corresponding to a census unit will fit one record using a specialized program.

Entry 1 usually performs the same program, but the output tape has another name. Bit 23 is set the whole time. It may be used only when it is set in the program to identify the start of the program.

* The segment may be omitted

Entry 2 (*optional*) is used before the running of the program in order to input some parameters into the standard area.

Program segments

a) *Program name/priority.*
It is written as in PLAN

```
1                    16
* PROGRAM            priority  name
```

b) *Peripherals*

```
1                    16
* PERIPHERALS  A, B, C, D...
```

where A, B, C, D ... are the symbolic names of the peripherals used by the programmer in PLAN segments.

Magnetic tapes, card reader and the line printer used by the standard program have program numbers: MT 14, MT 15, CRO and LPO and cannot be used by the programmer.

This directive is optional and generates the directives # PERIPHERAL and # CMODE in PLAN.

c) *Directives defining areas:*

```
1
* DIRECT AREAS
* INDIRECT AREAS
```

They are similar to # LOWER and # UPPER from PLAN and contain areas longer than one word and preset constants. The structure is the same as in PLAN.

d) *Description of the input file on magnetic tape*

```
1                        16
* INPUT MT               NAME0: XXXX XXXX XXXX
                         NAMEI: XXXX XXXX XXXX
                         DESCRIPTION: A
                             ed/name
                             — —  —  —  —
where                        — —  —  —  —
```

NAME0 = the twelve symbols occuring after |:| represent the name of the input file from the entry point 0.

NAME1 = the twelve symbols occuring after |:| represent the name of the input file from the entry point 1.

If NAME1 is missing at input 1 the input is considered to be on cards. The following symbols have been used:
e = number of decimal characters from the input record structure,
d = conversion characteristic.
B = ignores e characters — the instruction has the reduced form ed.

N = converts e characters in binary form and stores them into location "name". The form of the instruction is ed/name.

A = transfers e characters starting with the character whose location is specified at "name" The form of the instruction is: ed/name·k

If the binary conversion is not possible the location "name" will be # 77777777.

e) *Description of the input file on punched cards.*

```
1                                16
* INPUT PC                       DESCRIPTION: A
                                 ed/name
                                 — — — — — —
                                 — — — — — —
```

where

A = number of characters read from the card. The other specifications have the same sense as in d).

f) *Description of the output file on magnetic tape*

```
1                                16
* OUTPUT MT                      NAME0: XXXXXXXXXXXXX
                                 NAME1: XXXXXXXXXXXXX
                                 DESCRIPTION: A
                                 ed/name
                                 — — — — —
                                 — — — — —
```

where

e = number of characters in external form,

d = storage type.

The storage type may assume two values:

d = N, stored in binary form. *e* characters are transfered starting with the character (4-*e*) of the location "name" (*e* should be $\leqslant 4$).

d = A, stored in characters. *e* characters are transferred from the character having the address "name" *k*.

g) *Description of the output file on the line printer.*

```
1                                16
* OUTPUT LP                      TITLE: B
                                 SP: A
                                 XXXX..
                                 XXXX..
                                 XXXX..   max. 4 × 40 characters
                                 .....
                                 SP: A
                                 XXXX..
                                 XXXX..   max. 4 × 40 characters
                                 XXXX..
                                 .....
                                 CONTENTS: C
                                 SP: A
                                 LIST OF ERROR MESSAGES: D, name, K
```

B = number of spaces on a page,

A = $\begin{cases} 0,\ 1,\ 2 \rightarrow \text{number of spaces between two lines,} \\ \text{CH: X where } 1 \leqslant X \leqslant 7 \text{ throw to channel X of the control loop,} \end{cases}$

C = description of the spacing within the record,

D = the first character of the error message on the print-out sheet.

Name = the symbolic name of the error switches area. In this case a binary number is printed automatically; this number is 0 for each error indicator if the indicator is 0, and 1 whenever the indicator is not 0. The error indicators can be addressed by " name V", where V is the switch number. The numbering is from 1 to K, one by one. A new symbolic name is considered for $V > K$.

The main instructions are given in the Appendix.

3. THE TABULATING AND EDITING SYSTEM

Purpose of the system

This system is designed for statistical table processing starting from a given archive record in the following two different cases:

a) in order to obtain relative small, isolated tables if required; i.e. the considered archive is an information source for tables occurring more or less incidentally;

b) in order to obtain groups of tables, more complicated from the editing requirement point of view (e.g. censuses).

The general structure of the program system

The system will include a group of basic programs written for processing tables mentioned above at a). An additional group of auxiliary programs will meet the demands of b).

The main configuration includes a table generator (TG). The table generator is a compiler which accepts at input a source program and generates in PLAN an object program which, afterwards, will be compiled by # XPLJ. (Fig. 1). The object program (and the source description) has various possibilities especially from the table tabulation point of view. Its printing and computation facilities will be rather restricted. It will generate one or more table tapes and/or a print-out sheet having reduced editing possibilities.

The group of programs from the auxiliary configuration permits:

— the re-ordering of tables and the combining of tabulated tables;
— additional computations of the tabulated tables, concerning:
  — percentages;
  — additional rows or columns;
  — elaboration of mean tables, sums of squares, etc;
— the programs creating text tapes allow to obtain tapes containing texts and editing specifications for tables situated in table tapes; these programs receive at input parameter cards and data cards; it is also possible

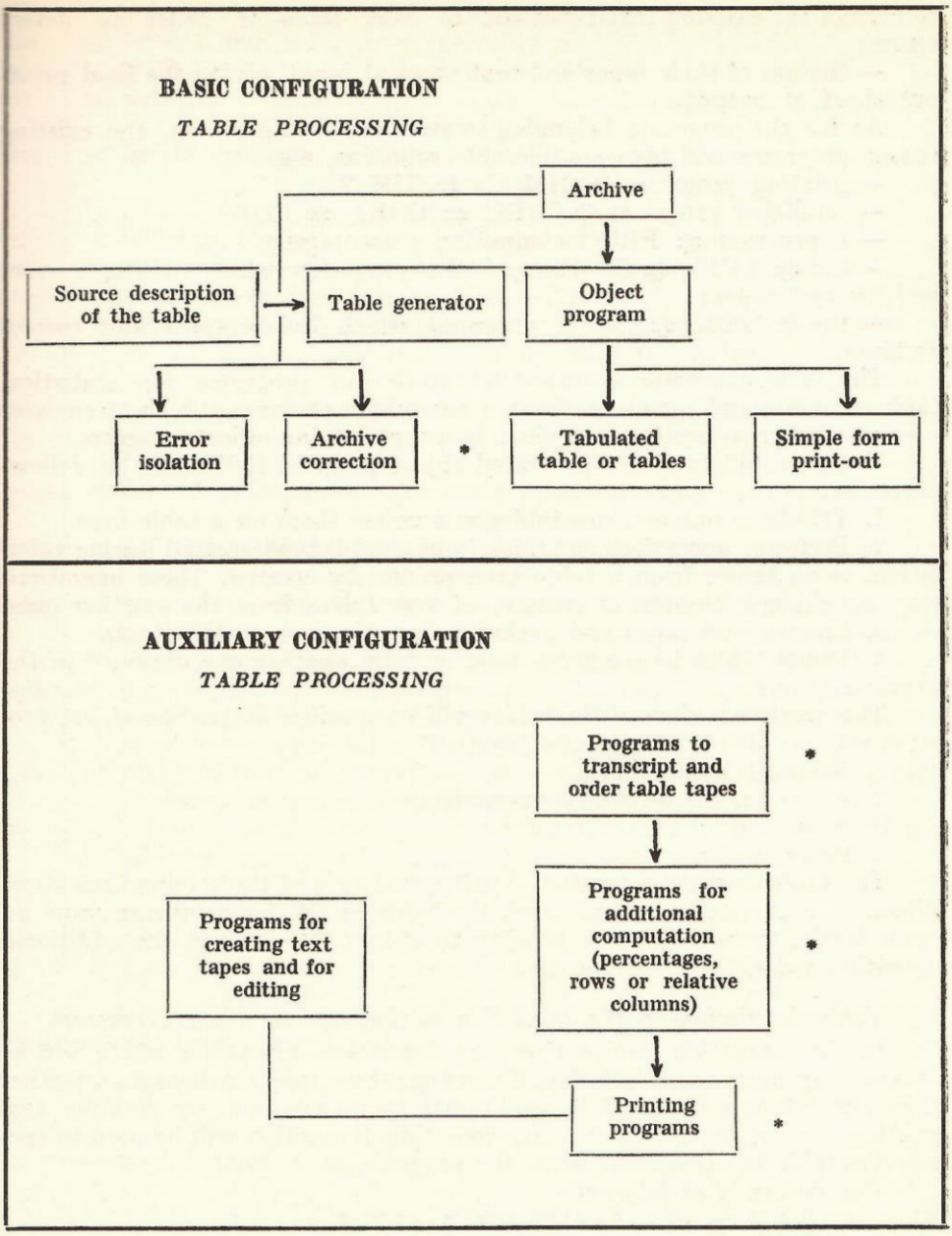


Fig. 1

NOTE. The programs marked by * can be obtained by slight modifications of existing programs.

to correct the existing text tapes and to edit them in order to detect errors;

— the use of table tapes and text tapes at input, giving the final print-out sheet at output.

As for the programs belonging to auxiliary configuration, the existing census programs will give considerable solutions, e.g.

— printing programs # GIM 1, # GIM 2

— totalizing programs # COR1, # COR2, # COR3

— a program # PROO computing percentages;

— the # LIBA, # COMB, # COP1 programs which create, correct and list text tapes;

— the # IZOL, # COAR programs which isolate cases and correct archives.

The table generator is meant to work out programs for statistical table making and printing. It is a compiling program which translates a source program written in a given language into an object program.

The possibilities of the generated object program (OP) are the following:

1. Tabulates one or more tables and writes them on a table tape.
2. Performs operations on tables, from a table tape created during tabulation, or on tables from a table tape previously created. These operations may be either arithmetic or creative of new tables from the existing ones.
3. Creates text tapes and performs operations on certain texts.
4. Prints tables from a given tape or from another one obtained in the previous stages.

The maximum sizes of the tables will be specified later. The object program will include the following sections:

1. Tabulation section.
2. A section for auxiliary operations.
3. A section which creates texts.
4. Print section.

The tables can be generated at different levels of the sorting indicators. When, at a particular sorting level, the table results by totalizing some inferior levels, there exists a possibility to obtain this level in the additional operation stage.

Table description in the tabulation section of the source program

In the tabulation section there are described only tables which will be obtained by means of tabulation. Therefore, these tables will contain neither rows nor columns obtained by additional operations (e.g. by dividing two existing rows of the table). Nevertheless, this description will be used to specify the table in other sections of the program.

The syntax is as follows:

	< digit > : : = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
	< letter > : : = A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
as	< alphanumerical symbol > : : = < letter > | < digit >
id	< identifier > : : = < letter > | < identifier > | < alphanumerical symbol >

inn	< indicator name > : : = < identifier >
tbn	< table name > : : = < identifier >
tol	< total list > : : = < digit > | < total list > + < digit >
inv	< indicator value > : : = < digit > | < digit >: < digit > | < total list >
linv	< list of indicator value > : : = < indicator value > | < list of indicator values > < indicator value >
inr	< indicator range > : : = (< list of indicator values >)
vin	< valuated indicator >: : = < indicator name > | < indicator range >
qtin	< quantitative indicator >: : = < digit > + < indicator name >
lqtin	< list of quantitative indicators > : : = < quantitative indicators > < list of quantitative indicators > < quantitative indicators >
qth	< quantitative hierarchy >: : = < list of quantitative indicators >
qlh	< qualitative hierarchy > = < list of qualitative hierarchy elements >
lqthe	< list of quantitative hierarchy elements >: : = < quantitative hierarchy elements > |
lqthe	< list of qualitative hierarchy elements >, < qualitative hierarchy elements >
qlhe	< qualitative hierarchy element >: : = < valuated indicator > | < qualitative hierarchy > | < juxtaposition of qualitative hierarchies >
ljqlh	< list of juxtapositions of qualitative hierarchies >: : < qualitative hierarchy elements > | < list of juxtapositions of qualitative hierarchies > | < qualitative hierarchy element >
jqlh	< juxtaposition of qualitative hierarchies >: : = [< list of juxtapositions of qualitative hierarchies >]
loe	< lay-out element >: : = [< qualitative hierarchy >, < quantitative hierarchy >]
	< lay-out list > : : = < lay-out elements > | < lay-out list > < lay-out element >
tlo	< table lay-out >: : = [< lay-out list >]
td	< table description >: : = < table name > | < table lay-out >

Fig. 2 represents the syntactical decomposition of a table description in the generator language.

Fig. 3 gives the corresponding table structure.

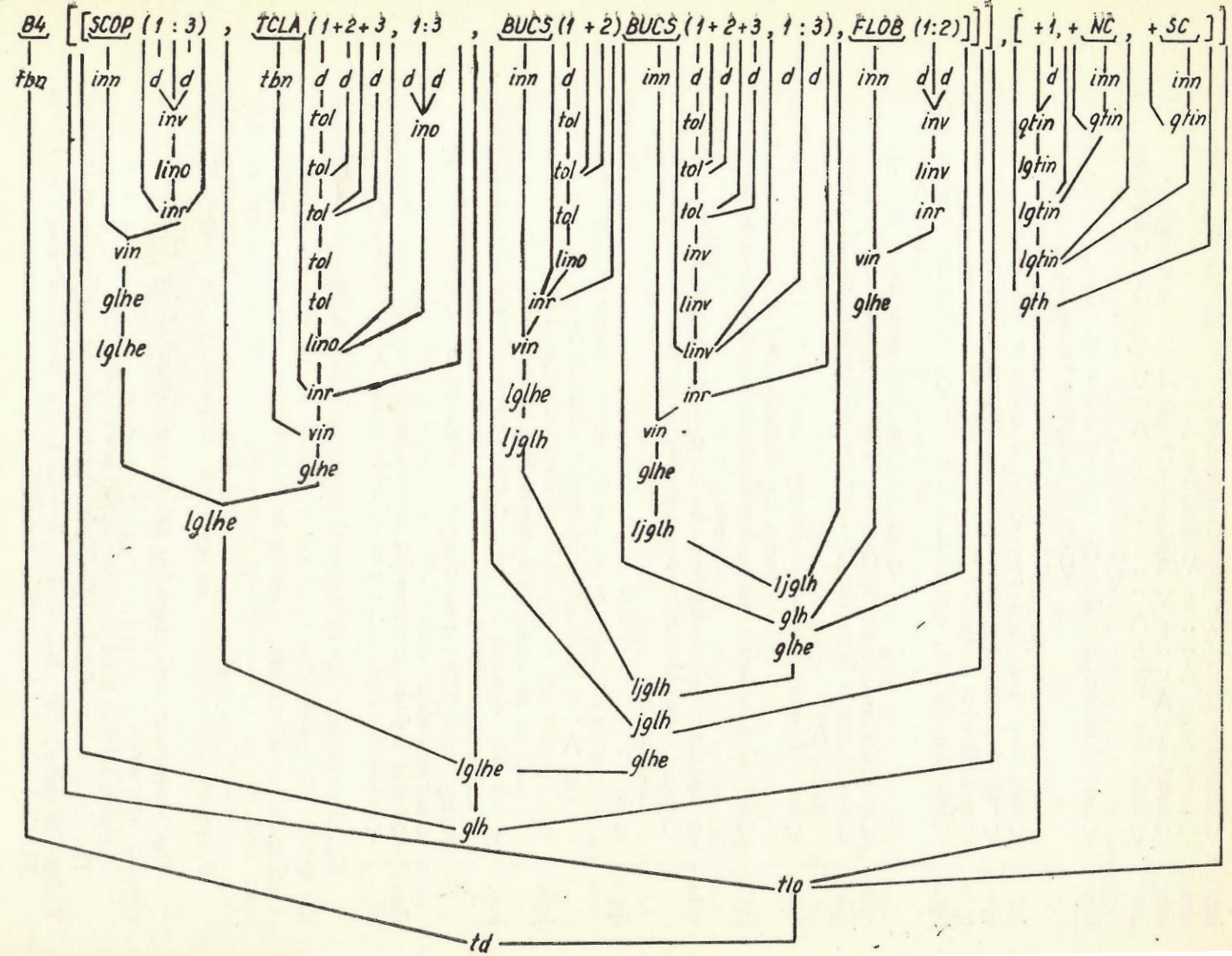
Operation description in the operation sections of the source program

The operations performed on tables are described by means of some specific operators.

Operator SUM is used to obtain total rows (columns).

The description of table TAB2 in the tabulation section is as follows:

TAB2 [[SCOP (1 : 3), TCLA (1 + 2 + 3, 1 : 3), BUCS (1 + 2 + 3, 1 : 3), FLDB(1 : 2)], [+1, + NC. + SC)]].


B4
[[SCOP (1 : 3) , TCLA (1+2+3 , 1:3 , BUCS (1 + 2) BUCS (1+2+3 , 1 : 3) , FLOB (1:2)]] , [+1, + NC , + SC]
tbn
inn
d
inv
lino
inr
vin
glhe
lglhe
tol
lino
ino
linv
linr
ljglh
glh
tlo
td

SCOP	TCLA	BUCS	BUCS	FLOB	
1	1+2+3				+ 1
					+ NC
					+ SC
			1+2+3	1	+ 1
					+ NC
					+ SC
				2	+ 1
					+ NC
					+ SC
			1	1	+ 1
					+ NC
					+ SC
				2	+ 1
					+ NC
					+ SC
			2	1	+ 1
					+ SC
				2	+ 1
					+ NC
					+ SC
			3	1	+ 1
					+ NC
				2	+ 1
					+ NC
					+ SC
	1				
	2				
	3				
2					
3					

Fig. 3

NOTE

In order to simplify the problem some boxes of the table are empty. The lay-out for TCLA— 1, 2, 3, is the same as for TCLA= 1+ + 2 + 3. In the same way the lay-out for SCOP = 2 and SCOP = 3 is the same as the lay-out for SCOP1.

If in this table it is necessary to obtain a total section resulting from TCLA = 1 and TCLA = 2 the sum-total of the sections BUCS = 1 and BUCS = 2 the resulting table TAB3 we shall write:

TAB3 [[SCOP(1 : 3), TCLA (1 + 2 + 3,1 + 2, 1 : 3), BUCS (1 + 2 + 3, 1 + + 2, 1 : 3),
FLDB (1 + 2, 1,2)], [+1, +NC, + SC] : = TAB2 [[SCOP(1 : 3),
TCLA (1 + 2 + 3, SUM(1,2), : 3), BUCS(1 + 2 + 3, SUM (1, 2), 1: 3), FLDB (SUM (1,2), 1 : 2], +1, +NC, + SC]]

Operator SUM can be applied only inside an indicator range. The created section can be set anywhere inside the corresponding indicator field.

Operator CONT generates a table starting from two given tables.

Example.

Let

TAB1 [[SCOP (1 : 3), TCLA (1 + 2 + 3, 1 : 3), BUCS (1 + 2 + 3 + 4)], [+1, + NC, + SC]]
TAB2 [[SCOP (1 : 3), TCLA (1 + 2 + 3, 1 : 3), BUCS (1 + 2 + 3, 1 : 3), FLDB (1, 2)], [+1, + NC, + SC]]
defined in the program tabulation section.

In the operation section we can write:

TAB4 [SCOP (1 : 3), TCLA (1 + 2 + 3, 1 : 3), [BUCS(1 + 2 + 3 + 4) [BUCS (1 + 2 + 3, 1 : 3),
FLDB (1 : 2)]], [+ 1 + NC, + SC] = CONT (TAB1, TAB2)

Operator SUBTAB creates subtables.

For example for TAB2 defined in the tabulation section we can write in the operation section:

TAB3 [[SCOP (1 + 2 + 3, 1 : 3), TCLA (1 : 3), BUCS (1 + 2 + 3)], [+ 1, + NC] : = SUBTAB (TAB2)

Operator JUXT juxtaposes two tables defined in the tabulation section or previously defined in the operation section.

For example:

TAB1 [[SCOP(1 : 3), TCLA (1 : 3), BUC(1 + 2)], [+ NC]]
TAB2 [[SCOP(1 : 3), TCLA(1), BUC(1)], [+SC)]]

we can write:

TAB3 [[[SCOP(1 : 3), TCLA(1 : 3, BUC(1 + 2)], [+ NC]] [SCOP(1 + 3), TCLA (1), BUC(1)], [+ SC]]] : = JUXT(TAB1, TAB2)

Operator PROC creates a percentage table starting from a previously created table.

For example: Let the table be:

TAB2[[SCOP (1 : 3), TCLA (1 : 3), BUCS(1 + 2 + 3), 1 :3)], [1 + NC ,+ SC]].

In order to create a percentage table the rows (columns) which percentages are computed must be specified:

TAB3 : = PROC(TAB2 [[SCOP(1 + 2 + 3), TCLA(1 : 3), BUCS(1 + 2 + 3, 1 : 3)], [+1, + NC, + SC]]

If in the different parts of a table we want to compute the percentages otherwise, the necessary subtables will be first created, and only afterwards the percentages will be performed.

Arithmetical operators ADD, SUBTRACT, MULT, DIV respectively add, subtract, multiply and divide two tables having the same number of lines and columns.

For example:

TAB4 : = DIV(TAB2, TAB3)
TAB6: = ADD(TA1, TA18)

A complete example of a source program

```
        1                    7
-------------------------------------------------------------
COMPILATION
                OBJECT PROGRAM                PROG95
                MT
INPUT           ARCHIVE, 1
            RAI 2(N, 1), COM2(B,2), SAT2(N, 1), REG2(N, 1)
            NMAP2(N,2), MED2(N, 1), NCLA2 (N, 2), SCOP2(N, 1)
            TCLA2(, 1), ANCON2(N, 2), MODC(N, 1), FPROC (N, 1)
            MPER(N, 1), MACO(N, 1), REE(N, 1), RGA2(N, 1)
            RAPA (N, 1), ICAN(N, 1), ICEN2(N, 1), ASC(N, 1)
            GAS (N, 1), GAGA(N, 1), FUC(N, 1), NIN(N, 2),
            GLOC (N, 1), CAPLOC (N, 2), NPIP( N, 2), NPSIP2(N, 2).
            NLTC (N, 2), NGTC (N, 2), NCADC (N, 2), NMGTC (N, 3)
            GLOC(N, 1), SCALC(N, 3), GLOC3(N, 1), NINNC(N, 2)
            SINNC(N, 2), GANCON(N, 1), GRAC(N, 1), GSUP (N, 1)
            CNET(N, 1), NLTCN(N, 2), NGTCN(N, 2), NMDTN(N, 2)
            NENNCN (N, 2), SINCN(N, 2), CNNIV(N, 2), GSCG(N, 1);
SORTING
    REG2, RAI2, COM2, SAT2;

    7          16        20
-------------------------------------------------------------
TABULATION CO7B   TABLE TAPE1, 200
           CO8X   TABLE TAPE12, 200
BELCRIT
           CO7B   SCOP2 = 1
           CO8X   SCOP2 = 2
ROWS       CO7B   [[TLCLA2(1,2), MPER2(3 + 4 + 5 + 6 + 9),
                  [[MPER2(2), MACO2(1 + 2 + 6, 1, 2, 6,)]
                  [MPER2(4), MACO2(1 + 2 + 5 + 7, 1, 2, 5,7)]
                  [MPER2(5)6), MACO2(1 + 2 + 3 + 4 + 5 + 7,
                      1, 2, 3, 4, 5, 7) ]
                  [MPER2(9), MACO2(9)]]], [+ 1]]
           CO8X   [[MED2(3, 4, 5), [SCOP2(1 + 2 + 4 + 5) [SCOP2(1)
                  TCLA2(1 + 2 + 3,1 : 3)]
                  [SCOP2(2, 4)], [+1, + NLTC, + NCALC, + SCALC
                  + NGTC, + NMGTC]], [[MED2(3, 4, 5), SCOP2(5)],
                  [+ 1]]]
COLUMNS    CO7B   [GANCON(1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 +
                  + 9 + 10 + 11 + 12 + 13 + 14 + 15 + 16 +17)]
```

```
CO8X                [[MPER2(1 + 2 + 3 + 4 + 5 + 6 + 7)MACO
                    (1 : 8)]
                    [MPER2(1 + 2 + 3 + 4 + 5 + 6), MACO (1: 7)],
                    [MPER2(7), MACO(8)]]
LEVEL       CO7B    REG2, COM (1 : 5)
            CO8X    REG2, RAI2
OPERATIONS
ROWS        CO8     [[MED2(3 + 4 + 5, 3, 4, 5), [SCOP2(1 + 2 +
                    + 4 + 5 [SCOP2(1) TCLA2
                    (1 + 2 + 3,1 : 3)] SCOP(2, 4)], [+ 1, + NLTC,
                    + NCALC, + SCALC
                    + NGTC, + NMGTC]], [[MED2(3, 4, 5), SCOP(5)],
                    [+ 1]]]: = CO8X [MED2 (SUM(3, 4, 5), SUM (3,4)]
FINISH
STOP
```

The program sections

COMPILATION section indicates the name and priority of the object program. MT indicates that the object program is already on magnetic tape.

INPUT section indicates the archive name (in 12 characters) and its generation number.

This section indicates also the archive structure by listing the indicators as follows:

< indicator name > (< indicator type >, < number of characters >)

SORTING section indicates the sorting keys of the input archive file.

TABULATION section describes the tapes on which the tables are printed. Optionally only one tape may be specified; afterwards this tape will record the tabulated tables in an adequate form.

The table description deals with rows and columns. It is important to note that the appearance of quantitative hierarchies in rows (columns) excludes their appearance in columns (rows). Finally, the LEVEL sub-section indicates the sorting indicators levels at which the table is produced.

OPERATION section indicates the operations performed in rows or columns. If we want to operate a sub-table we must first take off the sub-table by means of the SUB-TABLE operator.

The end of the program. We end the program whenever we give instructions #FINISH and #STOP.

Instruction OR

$$\text{OR}\quad \text{IF: name} = \left\{\begin{matrix}=\\ \#\end{matrix}\right\} \left\{\begin{matrix}\text{constant}\\ \text{limits}\\ \text{name}\end{matrix}\right\} \left[, \ \ldots \ \text{name} \left\{\begin{matrix}=\\ \#\end{matrix}\right\} \left\{\begin{matrix}\text{constant}\\ \text{limits}\\ \text{name}\end{matrix}\right\}\right]$$

$$\left[\text{TRUE: } \left[\text{name} = \left\{\begin{matrix}\text{constant}\\ \text{name}\end{matrix}\right\} \left[, \ \ldots \ \text{name} = \left\{\begin{matrix}\text{constant}\\ \text{name}\end{matrix}\right\}\right]\right], \ [\text{label}]\right]$$

$$\left[\text{FALSE: } \left[\text{name} = \left\{\begin{matrix}\text{constant}\\ \text{name}\end{matrix}\right\} \left[, \ \ldots \ \text{name} = \left\{\begin{matrix}\text{constant}\\ \text{name}\end{matrix}\right\}\right]\right], \ [\text{label}]\right]$$

Instruction AND

$$\text{AND IF: name} = \left\{\begin{matrix}=\\ \#\end{matrix}\right\} \left\{\begin{matrix}\text{constant}\\ \text{limits}\\ \text{name}\end{matrix}\right\} \left[, \ \ldots \ \text{name} \left\{\begin{matrix}=\\ \#\end{matrix}\right\} \left\{\begin{matrix}\text{constant}\\ \text{limits}\\ \text{name}\end{matrix}\right\}\right]$$

$$\left[\text{TRUE: } \left[\text{name} = \left\{\begin{matrix}\text{constant}\\ \text{name}\end{matrix}\right\} \left[, \ \ldots \ \text{name} \left\{\begin{matrix}\text{constant}\\ \text{name}\end{matrix}\right\}\right]\right], \ [\text{label}]\right]$$

$$\left[\text{FALSE: } \left[\text{name} = \left\{\begin{matrix}\text{constant}\\ \text{name}\end{matrix}\right\} \left[, \ \ldots \ \text{name} = \left\{\begin{matrix}\text{constant}\\ \text{name}\end{matrix}\right\}\right]\right], \ [\text{label}]\right]$$

Instruction LIM (limit control)

$$\text{LIM IF: name} = \left\{\begin{matrix}\text{constant 1}\\ \text{limits 1}\end{matrix}\right\} \left[, \ \ldots, \left\{\begin{matrix}\text{constant K}\\ \text{limits K}\end{matrix}\right\}\right]$$

[TRUE: [name = constant] [,] [label]]

[FALSE: [name = constant] [,] [label]]

Instruction COD (codify)

COD CODIFY: name

AFTER: name

$$\text{SO: constant} = \left\{\begin{matrix}\text{constant}\\ \text{limits}\end{matrix}\right\} \left[, \ \ldots \ \text{constant} = \left\{\begin{matrix}\text{constant}\\ \text{limits}\end{matrix}\right\}\right]$$

[OTHERWISE: [name = constant] [,] [label]]

Instruction COMP (comparison)

COMP COMPARE: { name / constant } [... { name] / constant }]

← n →

CV: { name / constant } [... { name / constant }]

n
←→

IF: { ≥ / ≤ / = / # }, [name = constant] [,] [label]

IF: { ≥ / ≤ / = / # }, [name = constant] [,] [label]

Instruction COR (correlate and correct)

COR FOR: name [, ... name]

MUST: name [, ... name]

OTHERWISE: name [, ... name]

LINE: { name / constant / limits }, { name / constant / limits }, { name / constant / limits } [, ... { name / constant / limits }]

[LINE: { name / constant / limits }, { name / constant / limits }, { name / constant / limits } [, ... { name / constant / limits }]]

[AND: [name = constant] [,] [label]]

Instruction LOOK FOR (Look for in the table)

LOOK FOR: name

IN: [name (table)]

[IS: [name] [,] [label] [,] [name = constant]

[NO: [label] [,] [name = constant]

Instruction FIND (find in the table)

FIND FIND IN: name (table)

STEP: constant

ADDRESS: { name / constant }

PUT IN: name (k) [, ... name (p)]

Instruction SC SUBTRACT (subtract)

SUB SUBTRACT: name, name

IN: name

Instruction ADD (add)

ADD ADD: name, name [, ... name]

IN: name

Instruction MULT (multiply)

MULT MULTIPLY: name, name [, ... name]

IN: name

Instruction DIV (divide)

DIV DIVIDE: name name

[READ: name]

[REST: name]

[INFINITE: label]

[ROUND]

Instruction JUMP (conditioned jump)

JUMP IF: name = { limits / constant / name } [, ... { limits / constant / name }] (n)

[THEN: label [, ... label] (n)]

[OTHERWISE: label]

GENERATEURS DE PROGRAMMES POUR LE CONTROLE LOGIQUE DES DONNEES STATISTIQUES ET LA CONSTRUCTION DE TABLEAUX STATISTIQUES

La programmation du traitement statistique pour un ordinateur se prête très bien à l'automation car, dans la plupart des cas, deux opérations sont nécessaires: (1) la préparation de plusieurs suites d'enregistrements et (2) leur utilisation pour élaborer des tableaux. C'est dans ce but qu'un langage d'ordre supérieur est élaboré dans le cadre de la Direction Centrale de Statistique pour faciliter ces opérations. A cette fin deux systèmes de générateurs de programmes sont élaborés: le premier crée des programmes de contrôle logique destinés à la formation d'archives correctes, de manière à ce que les tableaux de présentation ne puissent pas contenir d'incompatibilités; le second comprend un générateur de programmes pour les tableaux et un groupe de programmes auxiliaires pour les opérations supplémentaires sur les tableaux et leur impression.

Compte tenu des caractéristiques générales de pareils traitements et afin de simplifer des langages, certaines restrictions se sont imposées. Elles n'ont pas d'implications pratiques particulières, cependant elles ne peuvent pas être évitées.

Outre les programmes compris dans ce système, un système de formulaires sera également élaboré afin de définir et d'analyser le problème.

Les langages et les formulaires sont mis d'accord, de manière à ce que le passage de l'analyse du problème à l'élaboration des programmes-sources exige le moindre effort.

ГЕНЕРАТОР ПРОГРАММ ДЛЯ ЛОГИЧЕСКОЙ ПРОВЕРКИ ДАННЫХ И ПОСТРОЕНИЯ СТАТИСТИЧЕСКИХ ТАБЛИЦ

Программирование статистических обработок на электронном вычислителе может быть приспособлено с успехом к автоматизации поскольку, в большинстве случаев, должны быть произведены две операции: (1) подготовка рядов записи и (2) их использование к обработке таблиц. С этой целью, в рамках Центрального Статистического Управления Румынии разрабатывается язык высшего уровня, который облегчил бы эти операции.

Для этого вырабатываются две системы генераторов программ. Первая система генерирует программы логического контроля, предназначенные созданию качественных архивов, таким образом чтобы итоговые таблицы не могли бы содержать несовместности. Вторая система содержит генератор программ для таблиц и группу побочных программ для дополнительных операций по таблицам и их изданию.

Исходя из общих характеристик таких разработок и имея в виду упрощение языков, были необходимы некоторые ограничения не имеющие особого практического влияния и которые могут быть устранены.

Помимо программ, охваченных системой, будет разработана и система форм, необходимых для определения и анализа данного вопроса.

Языки и формы согласованы таким образом, чтобы переход от части, принадлежащей анализу вопроса, к разработке программ—источников, производился с минимальным усилием.