U N I V E R S I T Y O F L O N D O N

INSTITUTE OF COMPUTER SCIENCE

A D V A N C E D M V C M A N U A L

Volume II

by

ANDREW COLIN

October 1964

Preface

This provisional version of the MVC manual, Part II, has been published in response to a number of requests. It differs from the final version in three respects.

1) It does not incorporate any of the last-minute changes which will have to be made as the MVC system becomes fully developed;

2) It does not contain any description of the style and format of the output generated by the system;

3) The section on operating procedure (2.15) is highly provisional and is liable to change from week to week. Enquiries as to its current state should be addressed to the author.

It must be stressed that this manual does not form a complete handbook of the MVC system by itself. It should be read in conjunction with the previously published MVC Part I, to which it forms a continuation.

At present the author would be grateful to see all results produced by the MVC system, whether they are judged satisfactory or not.

From a copy preserved by Bill Williams and OCRed and re-formatted by Dik Leatherdale.

Original pagination has been preserved throughout but blank pages have been omitted. Modern fonts have been employed in the interests of readability. The contents list, index and internal references have been implemented as clickable links. **`Program and data text`** are differentiated by font from narrative.

Some limited correction of obvious errors has been performed and are shown in red, though trivial changes are not. Please report transcription errors to dik@leatherdale.net.

2.1 Some generalisations

In this section, we shall generalise some of the rules given in Part I of the manual.

2.1.1. Arithmetic Expressions.

Arithmetic expressions have a far wider field of application than that described in ELEMENTARY MVC. They can replace numbers and numerical variables in most contexts in an MVC specification. For example, they may be used in forming Boolean particles:-

```
derived binary;
BETTER AT SCIENCE:=((MATHS MARKS + PHYSICS MARKS) > (ENGLISH
MARKS + HISTORY MARKS));
```

Again, arithmetic expressions can be used in derived polylog definitions, e.g.,

```
derived polylog;
FINANCIAL STATUS:= (GROSS INCOME - TAX) <
                (500+100*NUMBER OF CHILDREN),
                (1500+100*NUMBER.OF CHILDREN),
                (1000000):
                POOR, MIDDLING, RICH;
```

Expressions are also useful in numerical table specification; for example:

```
correlation (NUMBER OF WET DAYS/LENGTH OF OBSERVATION),
                (cos(LATITUDE*3.14159265/180));
```

The brackets immediately surrounding the expressions in these examples are included for the sake of clarity. They are not syntactically necessary.

The one context in which the use expressions is specifically banned is that of raw definitions.

2.1.2. Boolean Expressions

Boolean expressions may replace Boolean particles in table specifications.

2.1.3. Polylog Answer Names,

In Part I, it was implied that the definition of every polylog variable had to include an explicit list of names for the possible answers. This is not in fact true. If the list of names is omitted, the system will automatically supply its own set of names, of the form

```
X(n)
```

where **X** is the name of the polylog variable, and **n**, an integer, shows which answer is meant. Thus, for the raw polylog definition (in the card convention)

```
IDEA,14/0,4;
```

the system will automatically supply the names

```
IDEA(1)
IDEA(2)
IDEA(3)
IDEA(4)
```

These names may be used as Boolean particles in just the same way as if they had been explicitly defined. It is legitimate to write such Boolean expressions as

```
IDEA(2) or IDEA(3)
```

2.1.4. Mixing the Character and Item Conventions.

With certain types of data, the 'character' and 'item' conventions, by themselves, may not be powerful enough to carry out a complete identification. For example, part of a record may be punched in the 'character' convention, with no separators between items of information, and another part may be punched as separate items. To cope with this eventuality, the facility of mixing the two conventions is provided. When using mixed conventions, the specification must start

```
read items;
```

Thereafter, 'item' definitions are given in their normal form; but in 'character' type definitions, the position of the starting character is given as

```
x/y
```

where x is the item number in which the character occurs, and y is the position of the character within this item. Thus, '`5/7`' in this context would indicate the seventh character in item 5.

To give an example: suppose that one of the items in a case record, mainly written in the item convention, is a 'case number', in which the first two digits are of special interest because they are used as a code for the subject's home district. The raw definitions for such a format might be:

```
read items;
raw numerical;
CASE NUMBER,1;
DISTRICT CODE,1/1,2;
..........
```

When using mixed conventions, care should be taken to ensure that any entities defined by 'character' definitions are completely contained within items. Thus if item number 3 in a record is known to contain — say — 7 characters, a definition such as

```
D1,3/5,4;
```

would always generate 'wrong' values for the numerical variable `D1`.

Sometimes it is useful to count characters not from the beginning, but from the end of some item. This can be done by a reference of the form

```
x/-y
```

which refers to the **y**'th character from the end of item (**x-1**). Thus, if in the sample above, the last 3 digits of the case number were a code for the subject's profession, we could write the definition

```
PROFESSION CODE, 2/-3,3;
```

2.1.5. Changing the 'natural order' of hole sites

As stated in part I, the 'natural order' of hole sites on a punched card is from top to bottom: that is, **U, L, 0,1,2,3,4,5,6,7,8,9** . Sometimes, particularly in the case of data which has been punched before the method of analysis was considered, this order is not suitable. If necessary, it can be changed by means of a special hole site order statement, of the typical form

```
hole site order 0,1,2,3,4,5,6,7,8,9,L,U;
```

The statement must mention each hole site once, in its required order.

The hole site order may be redefined several times in the course of a set of raw definitions. Each raw definition is governed by the nearest hole site order statement above it in the text.

2.2. Variable Data Format

One of the most difficult aspects of analysing large surveys is that of dealing with Questionnaires of variable format.

A variable format survey is one which the set of questions put to or answered for each subject is not invariant but changes from one subject to the next. There are two ways in which such a contingency can arise:

1) It is possible for the asking of a group of questions to be made conditional upon a particular answer being given to some other question. For example:

HAVE YOU A CAR?

If so, then: What is its make?
What is its year of manufacture?
What is its seating capacity?
Etc.

If not, then: Do you own any other means of transport?
Can you drive?
Etc.

2) It is possible for information on one case to include data on a variable number of 'sub-cases'. For example, a Questionnaire may include questions about the general characteristics of a household (which are answered only once per case), and further set of personal questions which must be answered once for each member of the household.

It is also, of course, possible for format variations as a result of combinations of these two causes.

To deal with these difficulties, it is clear that. devices of considerable power and sophistication must be used. The various facilities provided by the MVC system, when taken together, form a close approximation to a conventional programming language, and will be described in appropriate terms.

2.2.1. The value 'unknown'

The basis of the way in which variable format Questionnaires are dealt with is this:-

a) A variable is defined for every possible question in. the Questionnaire.

b) Questions which are not asked in any particular case give rise to the value 'unknown' for their corresponding variables.

c) During the construction of tables, facilities are provided which permit the sorting out of the known from the unknown values, and the use of the former only.

As an example of the concept, consider the transport Questionnaire given above. Three of the variables arising from this Questionnaire might be:

a) **HAS CAR** (binary);

b) **YEAR OF MAKE** (numerical);

and c) **HAS OTHER TRANSPORT** (binary);

Of these three, **HAS CAR**	would always have a value (yes or no);
YEAR OF MAKE	would only have a numerical value if a) was answered 'yes'. Otherwise it would have the value 'unknown'.
HAS OTHER TRANSPORT	would have a value (true or false) if a) was answered 'no'. Otherwise (since its question would not be asked) it would have the value 'unknown'.

In the construction of tables dealing with — say — the owners of cars, steps would be taken to include only those cases in which the relevant variables were not unknown.

The value 'unknown' is symbolically represented in an MVC text by the symbol **?** .

There are four ways in which variables may be assigned the value **?**. We list them in order of increasing importance.

1) Derived-numerical and binary variables may be explicitly assigned the value 'unknown" by definitions in which the right hand side is a **?**.

2) Derived variables whose defining expressions are for some reason not computable are also assigned the value **?**.

For example, if. **N2=N3**, then **N1** as-defined by-

```
N1:=1/(N2-N3);
```

will have the value **?**

Polylog variables are assigned the value if none of the possibilities listed in their defining expressions is true. For example, the derived polylog definition

```
P1:=NUMBER<1,2,3;
```

will set **P1 =?** if **NUMBER > 3.**

3) Incorrectly punched raw variables or those for which data is absent are given the value **?** if the case containing them is not rejected by a **check** statement in the survey specification.

4) Variables are automatically assigned the value 'unknown' if their definitions are not invoked at all. This is most important way of producing unknown variables. The facilities for skipping definitions are fully described in a subsequent section.

It should be noted that the specific answer 'unknown', actually given by a subject and punched as one of several possibilities does not give rise to the special value 'unknown'. for the corresponding variable. As far as the system is concerned, the subject's response to the question (i.e. that he does not know what to answer) is known.

2.2.2. The value 'unknown' in expressions

An MVC specification designed for a variables format survey will necessarily contains expressions in which some of the operands will sometimes have the value **?**. The rules governing the evaluation of these expressions are these:

1) If a polylog variable is unknown, then all the Boolean particles represented by its answer names are also unknown.

2) If an arithmetical expression contains an unknown numerical variable, then. the entire expression assumes the value 'unknown'. This value is also assigned to uncomputable expressions.

3) Unknown Boolean particles combine with other Boolean particles in the following way:

```
(F and ?) = (? and F) = F;
(T and ?) = (? and T) = ?;
(? and ?) = ?;
(F or ?) = (? or F) = ?;
(T or ?) = (? or T) = T;
(? or ?) = ?;
(not ?) = ?.
```

These rules are only formalisations of 'common sense'. For example. consider the expression

```
(MARRIED or (AGE > 21)).
```

If the particle **(AGE > 21)** is true, then the expression is obviously true irrespective of the value of **MARRIED**. It is true even if the value of **MARRIED** is unknown.

Variables can be tested for the value **?** by the existence operator, **E**. This operator is defined by:

E (Any Boolean or arithmetical expression of known value) = **T**

E (any Boolean or arithmetical expression whose value is **?**) = **F**

Thus:
```
E(?) = E(1/0) = E(sqrt(-1)) = F;
E(T) = E(F) = E(7) = E(E(?))= T.
```

2.2.1. A Simple Way of Coding Variable Format Questionnaires

One possible way of coding Questionnaires in which not all the questions are always asked is to represent the 'answers' to these questions by illegal punchings. For example, non-existent numerical answers can be represented by blank columns on punched cards or by non-decimal characters such as ? or * on punched tape. Provided that the specification does not

contain a check statement the corresponding variables will assume the value **?**.

Although this method is superficially attractive it is not recommended except for special cases. The chief objection is that check cannot be used (otherwise all cases with illegal punchings would be rejected from the survey); other objections are that the method is wasteful of computer time, and that it cannot be used with binary variables on cards.

The correct way of coding variable format Questionnaires is described in the next section.

2.2.4 Definition Skipping

The preferred method of treating Questionnaires of variable format consists of skipping the definitions of any variables to which the corresponding questions have not been asked. The method hinges on the possibility of using the values of variables already known to deduce which other variables are present, and which are absent. For example, consider the two questions

ARE YOU MARRIED?

IF SO, HOW MANY CHILDREN HAVE YOU?

Here it is possible to deduce the presence (or absence) of an answer to the second question from the answer given to the first.

2.2.4.1. Dynamic Interpretation of Definitions

The definitions of variables can be considered in two aspects; static and dynamic.

In the static aspect, which was. stressed in ELEMENTARY MVC, a definition is a statement that the information on a certain region of each case record (or the result of a certain computation on the values of other variables) is to be known by a specified name.: Considered dynamically, on the other hand, a definition is essentially an instruction to the computer to read a value from each case record (or to compute it) and to assign it to a named variable. It is thus similar in effect to an assignment statement in a conventional programming language.

The fact that definitions are interpreted dynamically implies that they must be invoked in some particular order. This order is normally that in which the definitions are written, but it may be varied by the mechanisms described below.

2.2.4.2. Labels and Jump Instructions.

Both labels and jump instructions are closely analogous to their counterparts in conventional programming languages.

Any statement in an MVC specification can be labelled by an integer in the range 0 to 255 and a colon; thus:

```
115: PLAYS CHESS, 51/2;
```

Labels are used by jump instructions, of which there are three basic forms:

1) **go to** <label>;

This is the unconditional jump.

2) if <Boolean expression> **go to** <label> ;

Control is transferred only if the Boolean expression is true.

3) unless <Boolean expression> **go to** <label> ;

Control is transferred if the expression is false.

The way in which labels and jump instructions are used to skip the definitions of non-existent variables is illustrated below. It is perhaps worth repeating here that any variable whose definition is skipped is assigned the value ?.

Consider the Questionnaire:

*Do you a) Own a television set;
or b) Hire a television set;
or c) Neither own nor hire a television set;

If you either own or hire a TV set, then:

Do you regularly watch a) The news
b) Panorama
c) Coronation St

What is the size of your screen?....

If you neither own nor hire a TV set, then:

Do you own a wireless set?

If so, do you listen regularly to

a) The news
b) Mrs. Dale's Diary

*Do you own a car?

In this questionnaire, the questions marked with an asterisk require answers from all subjects; the others do not.

In the possible set of definitions which follows, the card field information is included for the sake of completeness, but is irrelevant in this context.

It should be noted that unlike the definitions, the type declarations (such as **raw binary** or **derived numerical**) are not dynamically interpreted. Each definition is governed by the nearest declaration above it in the text.

```
     read cards;
     raw polylog;
     TV,10/0,3: OWNS, HIRES, NO TV;
     if NO TV go to 1;
     raw binary;
     TV NEWS,11/0;
     PANORAMA,12/0;
     CORONATION ST,13/0;
     raw numerical;
     SCREEN SIZE,14,2;
     unless HIRES go to 2;
     SHILLINGS,16,2;
     PENCE,18,2;
     derived numerical;
     RENTAL:= SHILLINGS + PENCE/12;
     go to 2;
     raw binary;
1:   WIRELESS,20/0;
     unless WIRELESS go to 2;
     RADIO NEWS,21/0;
     MRS DALES DIARY,22/0;
2:   CAR,23/0;
     ..........
```

2.3. Vectors of Variables

In Survey Analysis, one often comes across groups of variables which are identical in structure and similar in meaning. For example, a Questionnaire may ask for each of a students' 12 weekly marks throughout a term, or the ages of each of the several persons in one household.

In the MVC system, such groups of variables are conveniently handled as single entities called vectors. Vectors are closely analogous to 'arrays' in ALGOL or Fortran, or to 'main variables' in Mercury Autocode. They are not called arrays because they are restricted to one dimension only.

A vector of variable is assigned a name in the same way as any individual (non-vector) variable. When it is necessary to refer to any specific variable within a vector, a numerical subscript in square brackets is affixed to the general vector name. For example, if **WEEKLY MARK** and **AGE** were vector names, individual items within these vectors might be called

WEEKLY MARK [3] or **AGE [7]**

It is important to note that a vector consists of a group of variables which exist at the same time within one case, and correspond to a group of questions answered or every. subject. The successive values of one variable, which correspond to the different answers given to a particular question by different subjects, do not constitute a vector.

Vectors can be made up of variables of any of the three types, but naturally all the variables within any one vector must be of the same sort. Before a vector can be used, it must be declared by a statement of the form

vector <Vector name> [lower bound>:<upper bound>];

Here <Vector name> is the name of the vector being declared. And <lower bound> and <upper bound> are the lowest and highest sufficies to be used respectively. The lower bound mist be zero or possible.

Examples of vector declarations are:

```
vector DAILY TEMPERATURE [1:7];

vector BEST SUBJECT [1:20];
```

A vector whose bounds are [n:m] will contain m-n+1 members.

Depending on the context, vector names may be used either by themselves to indicate an entire group, or (as mentioned above) with numerical suffices to specify particular members of the group.

When the answer names of polylog vectors are mentioned in arithmetical expressions, they must also be followed by appropriate suffices. This rule also applies to the 'automatic' answers assigned by the system for polylogs whose answer names are not specified. Thus the Boolean particle

```
BEST SUBJECT (3) [6]
```

would be true if the value of the sixth variable in the polylog vector **BEST SUBJECT** was the third possible answer.

2.4. Indices

Indices are a device introduced to facilitate the definition and manipulation of vectors.

The indices are a set of 26 'special' variables, known by the names **A,B,C,.....Y,Z**. Indices always take integer values, and do not need to be specially declared.

The values taken by the indices are controlled by index setting statements. Each index setting statement consists of an index, the := sign., and a list of constants or arithmetical expressions separated by commas and terminated by a semicolon. Every time that an index setting statement is executed, the named index is assigned the next value in the list (or the integer nearest to this value). For example, the statement

```
P:= 1,3,15,46.9,3*2, sqrt(37),2,2,1;
```

will on successive executions, assign the nine consecutive values **1,3,15,47,6,6,2,2** and **1** to the index **P**.

Unlike a cycle setting in Mercury Autocode or a for statement in ALGOL, an MVC index setting statement does not in itself define a cycle or loop. Thus when several index setting statements follow one another, they are effectively executed in parallel.

The main use of indices is in compressing the definitions of vectors. The construction used is this:

1) the system word **for**
2) one or more index setting statements, which are executed in parallel
3) a set of definitions or other MVC text in which certain numerical constants are replaced by indices.
4) the system word **end** followed by a semicolon.

The effect of this construction is that the MVC text between for and **end** is invoked repeatedly until the index setting statement following it runs out of values. On each execution, the indices in the text are replaced by their current values.

To illustrate this concept, suppose that a punched card carries 12 scores, starting at column 10. The first six scores occupy two columns each, the next five 3 columns each, and the last one, one column. It is required to treat these scores as a vector, SCORE, with 12 members.

Without indices, the vector Would have to be defined by the sequence:

```
vector SCORE[1:12];
raw numerical;
SCORE[1],10,2;
SCORE[2],12.2;
SCORE[3],14,2;
SCORE[4],16,2;
SCORE[5],18,2;
SCORE[6],20,2;
SCORE[7],22,3;
SCORE[8],25,3;
SCORE[9],28,3;
SCORE[10],31,3;
SCORE[11],34,3;
SCORE[12],37,1;
```

By using the index facilities, these definitions con be shortened considerably. In the example below, **A** is used to index the members of the vector themselves; **B** is used to indicate the various starting columns, and **C** to show the number of columns.

```
vector SCORED [12];
raw numerical;
for  A:= 1,2,3,4,5,6,7,8,9,10,11,12;
     B:= 10,12,14,16,18,20,22,25,28,31,34,37;
     C:= 2,2,2,2,2,2,3,3,3,3,1;
     SCORE[A],B,C;
     end;
```

note that the index setting statements are executed in parallel. Thus, on successive execution **A,B,C** and will have the values (**1,10,2**),(**2,12,2**). and so on; and the definition **SCORE[A],B,C** will correspondingly be equivalent to the successive forms

```
SCORE[1],10,2;
SCORE[2],12,2;
etc.
```

Index setting statements in which the successive values are either constant or in arithmetical progression can be condensed by the following notations:

1) **a repeated b times** is equivalent to $\underbrace{\texttt{a,a,a,a,}\quad\texttt{a,a,}}_{\texttt{b} \text{ times}}$

2) **c step d until e** is equivalent to **c,(c+d), (c+2d),...**

The letters **a,b,c,d** and **e** may represent general expressions. It is for the user to ensure that in these constructions they form sensible combinations. The rules which must be observed are:

1) **b** must be positive

2) **(e-c)/d** must be a positive integer

Using these contractions, the index setting statements above may be written:

```
A:= 1 step 1 until 12;
B:= 10 step 2 until 22, 25 step 3 until 37;
C:= 2 repeated 6 times, 3 repeated 5 times, 1;
```

The format rules about. the use of indices are:

1) In raw definitions, indices may replace integers in any context except that of indicating the number of possible answers to a polylog question. (This must always be a constant, even for polylog vectors).

   In general, negative values for indices are permitted wherever they are meaningful. When an index is used as the row number in a card definition, the values **-2** and **-1** conventionally refer to holesites **U** and **L**, respectively.

2) In expressions, indices may be used just as though they were integer constants.

3) Neither indices nor index setting statements may under any circumstances be used inside table specifications.

4) Nesting of **for**-**end** cycles is permitted up to a depth of 6. This facility might be used in the case when a single cycle is insufficient to define a vector compactly. Suppose, for example, that a card contains 300 polylog answers, each with three possibilities. Each column, starting at column 6, contains four answers, occupying the fields U to **0**, **1** to 3, 4 to **6** and **7** to **9**, respectively. This example also illustrates the use of expressions in index setting statements. A possible way of defining the vector **RESPONSE** would be this:

```
vector RESPONSE[1:300];
     for C:= 6 step 1 until 80;
          for  D := (4 * C - 23) step 1 until (4 * C - 20);
               R:= -2 step 3 until 7;
               raw polylog;
               RESPONSE[D],C/R,3:YES,MAYBE,NO;
          end;
     end;
```

This sequence is, of course, equivalent to the definitions

```
RESPONSE[1],6/U,3;
RESPONSE[2],6/1,3;
RESPONSE[3],6/4,3;
RESPONSE[4],6/7,3;
RESPONSE[5],7/U,3;
RESPONSE[6],7/1,3;

.................
.................
RESPONSE[300],80/7,3;
```

5) In **goto** statements, indicies may be used in two ways: would be this:

a) They may be included in the Boolean expression contained in conditional jumps. It is legitimate, for example, to write

```
if (J ≠ 7) go to 68;
```

b) They may also be used to indicate the destination label symbolically. One may write, for example,

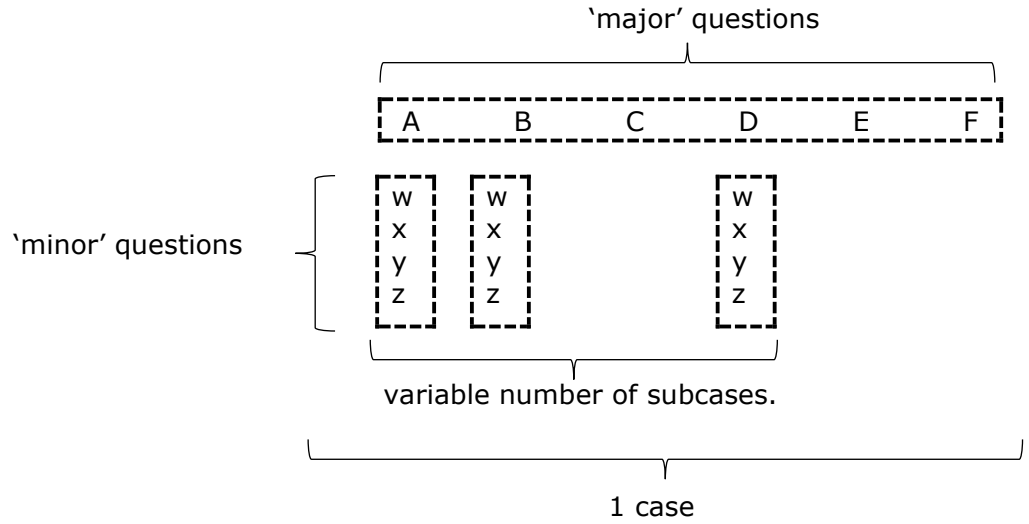
```
go to J;
```

This will have the effect of transferring control to the label whose numerical value is the same as the current value of **J**. Thus, if **J = 89**, '**go to J**;' Will have the same effect as '**go to 89**;'.

2.5. The Use of Vectors in Variable Format Data.

A considerable source of difficulty for most methods of Survey Analysis has been the existence of Surveys in which information on each case, or 'unit of population', is accompanied by data on each of a variable number of 'subcases'. For example, in a survey in which the unit is a household, each case record may include data on each of the children present in that household. Again, a survey on transport may ask the subject to list details of each of the journeys made during a particular period.

The data structure encountered in each such a survey may be diagrammatically represented thus:



Here, the 'major' questions refer to the case as a whole, and the 'minor' questions to each of the subcases. For example, A, B and C might be

'What is the total family income?'

'What is the family's type of dwelling?''

and 'How may children in the household?'

whereas w,x,y and z could be

'What is the age (of this child)?'

'What is the age (of this child)?'

'What type of school does this child attend?'

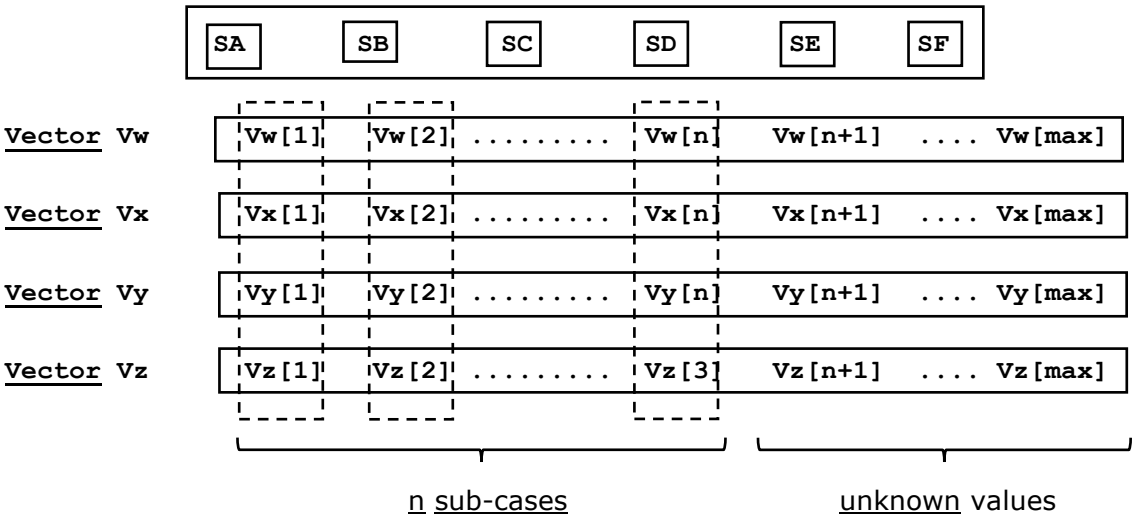
and 'Does this child own a bicycle?'

Each 'subcase' would comprise the information relating to one child.

In the MVC system this type of structure is handled by assembling corresponding variables in the several subcases into vectors. Thus, a general vector name might be '**OWNS BICYCLE**'; so that '**OWNS BICYCLE[1]**' would refer to the first subcase, and so on.

In declaring these vectors, the upper bound is made sufficiently high to allow for a reasonable maximum number of sub-cases (e.g.12 if the subcases are children in one family). The definitions are arranged so that only those members of the vectors which correspond to actual sub-cases of any particular case are assigned values, the other remaining unknown.

Thus, the way in which variables are imposed upon the data structure shown above is this:



Here, **SA**, **SB**, **SC**, **SD**, **SE** and **SF** are simple (non-vector) variables which correspond to A,B,C,D,E, and F; and the vectors **Vw**, **Vx**, **Vy** and **Vz** correspond to the groups of questions w,x,y and z. In a case which contains n sub-cases, only n members of each vector are known; the remainder take the value **?**. We shall illustrate this concept by considering a Questionnaire which (we suppose) has been put to school-teachers preparing classes for 'O' level examinations. The. Questionnaire requires details about each pupil, but the main subject of investigation is the teacher himself. We shall therefore consider all the information given by any one teacher to form a case, the data on the pupils comprising sub-cases.

Data relating to each teacher will be punched on several cards, one for the teacher himself, and one for each of the pupils. Each card will carry a four-digit case number in columns 1 to 4 (this being the same for a teacher and all his pupils), and a two-digit 'card-in-case' number in columns 5 and 6. This will .always be '**01**' for the teacher, '**02**' for the first pupil, '**03**' for the second, and so on.

The Questionnaire itself is this:

Case Number | 0 | 7 | 7 | 6 | Cols 1-4

Part 1: to be answered by the teacher himself (or herself)

Card in case number | 0 | 1 | Cols 5-6

1) | 5 | 1 | Cols 7-8

2)	What is your sex?	a)	MALE	✓ 9/0
		b)	FEMALE	9/1
3)	What group of subjects are you teaching?	a)	ENGLISH	10/0
		b)	MATHEMATICS	✓ 10/1
		c)	A 'SCIENCE' SUBJECT	10/2
		d)	A FOREIGN LANGUAGE	10/3
		e)	CLASSICS	10/4
		f)	HISTORY OR GEOGRAPHY	10/5
		g)	DIVINITY	10/6
		h)	COOKING OR NEEDLEWORK	10/7
		i)	ANY OTHER	10/8
4)	What are your qualifications?			
		a)	TRAINING COLLEGE	11/0
		b)	UNIVERSITY	✓ 11/1
		c)	UNIVERSITY AND T.T.C.	11/2
		d)	OTHER	11/3
		e)	NONE	11/4

5) How many years have you taught this subject? | 2 | 4 | Cols 12-13

6) How many pupils are you preparing for 'O' level in | 0 | 5 | Cols 14-15

6) in this subject this year

Part 2 To be answered for each pupil — On each card:

Pupil number		1	2	3	4	5				Not punched
Card in case number		02	03	04	05	06				Cols 5-6
Age of pupil										Cols 7-8
Sex of pupil:	GIRL		✓	✓		✓				9/0
	BOY	✓			✓					9/1
Was the Term's work	a) OUTSTANDING					✓				10/0
	b) GOOD		✓		✓					10/1
	c) AVERAGE	✓								10/2
	d) POOR			✓						10/3
Are this pupil's Chances of gaining A pass	a) ALMOST CERTAIN		✓		✓	✓				11/0
	b) DOUBTFUL	✓								11/1
	c) BAD			✓						11/2

In writing definitions for this questionnaire, we shall make use of indices and vectors, each corresponding to a group of similar variables taking one from each subcase.

```
read cards;
case number 1,4;
raw numerical;
TEACHERS AGE, 7,2;
YEARS TAUGHT,12,2;
NUMBER OF PUPILS,14,2;
raw polylog;
TEACHERS SEX,9/0,2: MALE,FEMALE;
SUBJECT,10/0,9:ENGLISH,MATHS,SCIENCE,LANGUAGE,
               CLASSICS, HIST OR GEOG, DIVINITY,
               DOMESTIC, OTHER SUBJECT;
QUALIFICATIONS,11/0,5: TTC,UNIVERSITY, UNIV AND TCC,
                                  OTHER QUAL, NONE;
comment: This completes the definitions of the major variables.
Now come the definitions for the subcases;
vector AGE OF PUPIL[1:30];
vector SEX OF PUPIL[1:30];
vector WORK OF PUPIL[1:30];
vector CHANCES OF PUPIL[1:30];
comment: These vectors allow up to 30 pupils per teacher;
for X:= 1 step 1 until NUMBER OF PUPILS;
    Y:= 2 step 1 until NUMBER OF PUPILS + 1;
      raw numerical;
      AGE OF PUPIL[X], Y.7,2;
      comment: Note use of index card number;
      raw polylog;
      SEX OF PUPIL[X], Y.9/0,2: GIRL,BOY;
      WORK OF PUPIL[X], Y.10/0,4:OUTSTANDING,GOOD,AVERAGE,POOR;
      CHANCES OF PUPIL[X], Y.11/0,3:CERTAIN,DOUBTFUL,BAD;
end;
```

Suppose that a particular teacher, having only 5 pupils, has answered the Questionnaire as shown. Then only the first five elements of each vector will be known, and the remaining 25 will be unknown. The values of some individual variables for this case will be:

```
SEX OF TEACHER = MALE;
NUMBER OF PUPILS = 5;
AGE OF PUPIL[1] = 15;
SEX OF PUPIL[2] = GIRL;
CHANCES OF PUPIL[5] = CERTAIN;
SEX OF PUPIL[6] = ?
AGE OF. PUPIL[30] = ?
```

In this example, we have taken advantage of the fact that the number of pupils in each case is explicitly given to make the index cycle repeat itself the exact number of times required. If the number of pupils had not been given, then the index setting statements would have had to be written as

```
X:= 1 step 1 until 30;
Y:= 2 step 1 until 31;
```

The portions of vectors which, in any one case, corresponded to non-existent sub-cases would have been assigned the values 'wrong' and results could only have been used in the absence of a '**check**' statement. (see Sections 1.11.1 and 2.2.1.)

2.6. Vectors in Expressions

In general, vector names may be freely used in expressions. Under normal circumstances, vector names will be accompanied by suffices in square brackets. Suffices may consist of arithmetical expressions of any complexity, and on evaluation are interpreted as being equivalent to the nearest integer. It is the user's responsibility to ensure that the values of suffix expressions remain within the range set by the bounds of the vector they refer to.

When vector names occur on the left, hand sides of definitions, then the form of suffix is strictly limited to an integer or a single index. No form of. suffix 'expression' is permitted.

For example, the form

```
AGE OF PUPIL[NUMBER OF PUPILS]
```

which refers to the age of the last pupil listed by the teacher, is permissible on the right hand side of a definition, but not on the left hand side.

To assist in the manipulation of vectors which contain an arbitrary number of unknown elements, a number of special 'vector functions' are available. All the vector functions take single unsuffixed vector names as arguments. They are listed below:

1) Functions applicable to all vectors:

number(<vector name>) = the number of known elements in the vector.

2) Functions applicable to numerical vectors: **sum**(<vector name>) = the sum of the known elements.

product(<vector name>) = the product of the known elements

In addition, the functions **max**, **min**, **whichmax** and **whichmin** may take single vector names as arguments.

3) Functions applicable: to binary vectors;-or polylog answer vectors

some(<vector name>)= **T** if any of the known members of the vector have the value **T**; otherwise **F**.

all(<vector name>) = **T** if all the known members have the value **T**; otherwise **F**.

All the vector functions except **number** generate the value **?** if none of the elements in the argument vector is known.

To illustrate these functions, we shall give some derived definitions which might be included in the school-teacher survey.

```
derived numerical;
AVERAGE AGE OF CLASS: = add(AGE OF PUPIL)/NUMBER OF PUPILS;
derived binary;
TEACHER HAS OUTSTANDING PUPILS:= some(OUTSTANDING);
derived polylog;
CLASS TYPE: = some(GIRL) and some(BOY), all (GIRL), all(BOY):
                                        MIXED, BOYS ONLY, GIRLS ONLY;
```

2.7. Programming in MVC

In some surveys, it may happen that the transformations to be applied to raw variables to obtain the desired derived variables are so complicated that they cannot be specified by single expressions. It is quite permissible to write a 'program' in MVC in which derived definitions are treated as 'statements' in a conventional programming language. Jump statements (i.e., those beginning with **if**, **unless** and **goto**) may transfer control either forwards or backwards. If necessary, the same variable may occur on the left of two or more definitions.

To illustrate this idea, suppose that in a survey there are two raw numerical variables, **EN** and **AR** where **AR** is always a positive integer. It is required to form the derived variable DONG according to the formula

$$\mathbf{DONG = 1 + EN + EN^2/2 + EN^3/3 + EN^4/4 + ... \ EN^{AR}/AR.}$$

This formula cannot be written as a single MVC expression because the number of terms is variable; but it can be programmed as follows:*

```
DONG: = 1;
for J := 1 step 1 until AR;
      DONG : = DONG + (EN|J)/J;
end;
```

If used in this way, MVC offers most of the facilities associated with conventional programming languages. (Notable exceptions are block structure and recursive procedures).

A further facility, introduced primarily for test purposes in developing the system, but useful, for example, in drawing the user's attention to extreme, or unusual values of variables, is the '**print**' statement.

This takes the form:

print <variable name>;

or **print** <vector name> [<integer or index suffix>];

The effect of this statement is to print the name of the variable, followed by its value. Printing is done before any tabulation takes place. The examples below refer to the school-teacher survey:

a)
```
print TEACHERS SEX;
```

This would cause the sex of each teacher to be printed in the form

```
TEACHERS SEX = MALE
```

b)
```
if (TEACHERS AGE < 70) go to 5
print TEACHERS AGE;
5: ------
```

This group of statements would print the age of each teacher aged 70 or more.

---

* This is not an optimum program in the sense of efficiency; it was chosen. for its relative clarity.

c)
```
for A := 1 step 1 until NUMBER OF PUPILS;
unless OUTSTANDING[A] go to 7;
print WORK OF PUPIL[A];
7: end;
```

This would print, for each case, the pupils who were classed as outstanding:

```
WORK OF PUPIL[1] = OUTSTANDING;
WORK OF PUPIL[17] = OUTSTANDING;
etc.,
```

When using the **print** statement, there is no need to specify explicit printing of the case number so as to identify the subject of the case; as we shall explain later, the case numbers of all cases dealt with are printed automatically.

2.8. Further Checking Facilities.

An aspect of survey analysis which is of fundamental importance is that of ensuring that data is presented to the computer in the right way. It is not difficult for an operator to feed in a reel of punched tape correctly, but it is singularly easy for him to make mistakes when putting through a stack of punched cards. A harassed and inexperienced operator can put part of a batch of cards through a card reader twice; he can put them in backwards or upside-down; or he may drop them so that the original order is destroyed. Any such occurrence which took place without the user's knowledge would have catastrophic results, since it would completely upset the results of the survey.

The MVC system includes a number of checks to avoid. this contingency. Some of them are automatic, and others have to be explicitly programmed in the specification.

2.8.1. Checks on Case Numbers

The system requires that the case numbers of consecutive cases be in ascending order. It will print a warning message and reject any case whose case number is not higher than the highest case number accepted so far. This ensures that no case is used twice, but demands that data on paper tape be punched in the right order, and that data punched on cards be pre-sorted on a card sorter before being used.

As each case is analysed, its case number is printed. Thus any messages about a case, such as reasons for its rejection or information produced by '**print**' statement, can be identified by the case number immediately above them. When the reading of data is complete, the list of case numbers produced can be scanned to ensure that none have been missed out.

2.8.2. Checks on Cases Each Containing Several Cards.

When reading data in which each case is represented by several cards, the system assembles each case record from all consecutive cards which have the same case number. The first of these is considered to be card 1, the second card 2, and so on. For the correct identification of data, it is important that the cards within a case be in the right order; but the system does not automatically check that this is so. (It would have been impractical to include such a facility, because in many applications the card-in-case numbers within a case are not consecutive and do not occupy the same positions on all the cards.

A check for correct order of cards within a case can be explicitly specified by defining raw variables corresponding to the card-in-case numbers, and checking that they take their expected-values. One might write for example:

```
CARD IN CASE NUMBER 1, 80,1;
CARD IN CASE NUMBER 2, 2.4,1;
CARD IN CASE NUMBER 3, 3.79,2;
accept (CARD IN CASE NUMBER 1 = 1);
accept (CARD IN CASE NUMBER 2 = 7);
accept (CARD IN CASE NUMBER 3 = 12);
```

This example assumes that the first card of each case-has a 1 in column 80; the second, a 7 in column 4; and the third, the digits 12 in columns 79 and 80. (This example, although it seems unlikely; is representative of several instances known to the author.)

If the card-in-case numbers are regular, then one can considerably shorten the written form of the check by using indices and vectors; in the schoolteacher survey we might have put:

```
raw numerical;
vector CARD IN CASE NUMBER N[1:31];
for J: = 1 step 1 until NUMBER OF PUPILS + 1;
     CARD IN CASE NUMBER N, J.5,2;
     accept (CARD IN CASE NUMBER N[J] = J);
end;
```

Sometimes it is possible to deduce either from prior knowledge or from the values of certain variables what the total number of cards in each case should be. Where this is so, the actual total can be compared with the expected total means of the '**total cards**' facility. This is specified by the statement

```
total cards <arithmetical expressions>;
```

This statement causes rejection-of cases for which the two totals do not match. Example of its use are:

```
total cards 5;
```

or (in the case of the teacher survey);

```
total cards NUMBER OF PUPILS + 1;
```

A total cards statement involving variables must be placed after the definitions of these variables.

2.9. Weights

In certain types of survey it is sometimes necessary. to attach different 'weights', or significances to different cases. For example, the answers given by one particular group of subjects might be considered of greater or lesser moment than those given by other subjects. Weights differing from 1 may be assigned to chosen groups of cases by the 'weight' facility, whereby any case having a weight of x (where x is not necessarily integral) produces exactly the same effect as x cases of weight 1.

The table headings of any tables in which cases are to have varying weights may contain one or more statements of the following form:

weight <Boolean expression>,<arithmentical expression>

for example,

```
weight (AGE>50), 2;
```

or

```
weight ATTENDED UNIVERSITY, (5 - CLASS GAINED) | 2;
```

The effect of such statements is to assign weights equal to the values of the arithmetical expressions to any cases for which the Boolean expressions are true, leaving all other cases with a weight of 1. Thus, the first example above would assign a weight of two to people over 50; and the second example would select those who had at attended University, and give them weights computed from the clases they had gained, as follows:

Class	Weight
I	16
II	9
III	4
IV	1
FAIL	0

Wherever a table heading contains several statements, they are evaluated in their sequence, and the weight actually assigned to any case is that specified by the last weight statement found to apply, Thus, the sequence

```
weight MALE, 2;
weight MALE and (AGE > 40),4;
```

would not have the same effect as

```
weight MALE.and (AGE>40),4;
weight MALE, 2;
```

Since **MALE** and **(AGE>40)** is a subclass of **MALE**, the first statement in the second pair above is effectively 'masked' by the second.

A weight statement is only effective in the table specification in which it is included. To give a case a constant weight throughout a complete set of tabulations, the **global weight** facility should be used. This statement is exactly similar to the weight facility, except that the statement begins with **global weight**, and have effect throughout all tables, except where they are specifically over-ridden by local **weight** statements. Since global weight statements do not apply to any specific table, they are placed outside (usually before) any table specifications.

2.10. More About Numerical Specifiers

2.10.1. The Moment Facility

In many statistical investigations, the average and the standard deviation of sets of numerical variables are often required together. Both these statistics can be obtained simultaneously by the **moment** facility.

Thus, where the table specification

```
mean X1, X2, X3;
stdev X1, X2, X3;
```

would have produced

```
MEAN
X1          19.7
X2          39.2
X3          4.3
STDEV
X1          1.32
X2          5.16
X3          0.87
```

the specification

```
moment X1 X2,X3;
```

will generate

```
MOMENT
X1          19.7
STDEV        1.32
X2          39.2
STDEV        5.16
X3           4.3
STDEV        0.87
```

2.10.2. Correlation Matrices

The set of correlation coefficients (See section 1.15) generated by a statement of the form

```
correlate N1, N2, N3, ............;
```

is, of course, a triangular correlation matrix, since it contains the correlation coefficients between every possible pair of variables. Sometimes it is more convenient to specify a rectangular matrix, which contains the correlations between each of the variables in one group and each of the variables in another group, only. This can be done by inserting the system word **by** between the two groups in the normal **correlate** statement.

Thus, the statement

```
correlate Y1,Y2,Y3,Y4,Y5,Y6;
```

would generate the triangular matrix

will generate

CORRELATION	Y1	Y2	Y3	Y4	Y5
Y2	*				
Y3	*	*			
Y4	*	*	*		
Y5	*	*	*	*	
Y6	*	*	*	*	*

but the statement

```
correlate Y1, Y2, Y3, Y4 by Y5, Y6;
```

would produce the rectangular matrix

CORRELATION	Y1	Y2	Y3	Y4
Y5	*	*	*	*
Y6	*	*	*	*

In both these illustrations, the * represents a correlation coefficient between the variables named above it and to its left.

At this point we shall repeat the rules that expressions may be used instead of numerical and binary variables in table specifications, but that any use of indices is barred.

2.11. Unknown Quantities in Table Specifications

In the course of analysis of a survey it may arise, either by accident or design, that certain quantities mentioned in a table specification are unknown for some specific case. When this happens, that case is not used to increment that table. The effect is the same as if the case had been excluded by an entry criterion.

One consequence of the rule is that it is very easy to specify tables referring to a subset of the survey sample for which certain 'special' questions have been answered. For example, suppose that we require a tabulation of the television watching habits of the people who answered the questionnaire on P(2-10). We could simply put:

```
title 'TV WATCHING HABITS';
count TV NEWS, PANORAMA, CORONATION ST;
finish;
```

The resultant table would only refer to those people who had TV sets and had answered the relevant questions. Other people would automatically be excluded because the values of the relevant variables in their cases would be 'unknown'. It is worth noting, however, that unless the cases to be included in such a table form a substantial majority of the survey sample, it is more efficient, from the point of view of computer usage and cost, to include a specific entry criterion; e.g.,

```
title 'TV WATCHING HABITS';
include OWNS or HIRES;
count TV NEWS, PANORAMA, CORONATION ST;
finish;
```

Another application of the general rule occurs when the data is known to be of poor quality, with many randomly scattered unknown variables. To extract the maximum amount of information from such data, the MVC text should contain a large number of very. short table specifications, rather than a small number of large ones; this will obviously lessen the probability of any one table specification referring to an unknown variable.

2.12. The Use of Vector Names in Table Specifications

In general, the items mentioned in table specifications are single-valued, in the same sense that they refer to single numerical, binary or polylog quantities. Vector names may be used freely, provided that they conform to this rule. Thus, they will normally be followed by suffices or be the arguments of single-valued functions such as **max** or **some**. For example, one could legally write (borrowing from the example in 2.5.)

```
     count CHANCES OF PUPIL [1];
or   mean max (AGE OF PUPIL), min(AGE OF PUPIL,
     max(AGE OF PUPIL) - max(AGE OF PUPIL);.
```

(this statement requests the average maximum age, the average minimum age, and the average range of ages of the pupils in each of the classes under consideration).

When specifying numerical or binary quantities by expressions, the suffices of vectors may themselves be arithmetical expressions of any complexity. For example, the Boolean particle

```
GIRL[whichmin (AGE OF PUPIL)]
```

would be true if the youngest pupil in any class was a girl.

On the other hand, it should be remembered that polylog question names are not expressions. The suffix attached to a polylog question vector name must always be an integer. The form

```
SEX OF PUPIL[whichmin (AGE OF PUPIL)]
```

is barred.

This restriction should not create any real difficulty, because in nearly all contexts in table specifications, polylog questions are synonymous to the list of their names, and may be replaced by them.

2.12.1. The '**list**' Construction

As stated previously, the use of indices in table specifications is barred. Nevertheless, there is a shorthand construction by which all members of a vector may be specified. If, elsewhere in the specification we have the declaration

```
vector NAME [m:n];
```

(where **NAME** is the name of some vector), then inside a table specification, the form

```
list NAME
```

is equivalent to the sequence'

```
NAME[m], NAME[m+1], ........ NAME[n-1], NAME [n]
```

This construction is also valid for the answer names of polylog vectors.

If the **list** construction is applied to a vector in which not all the elements are known the equivalent expanded list will necessarily contain references to unknown variables, and cause that case to be passed over in tabulation. This means that the **list** construction is only of practical use when all the members of a vector are known to be known.

2.12.2. Sub-case Tabulations

This facility is intended for surveys in which each case contains a variable number of subcases, which are represented by vectors in the manner described in 2.5. It forms a means of promoting each subcase into a complete 'case' on its own, while still retaining access to the major variables referring to its parent case. The facility is powerful and requires considerable care in its use. We shall illustrate it by referring to the scholastic survey mentioned in 2.5.

Any case which includes sub-cases contains a number of vectors, all with the same bounds. (Our example contains the vectors **AGE OF PUPIL**, **SEX OF PUPIL** and **CHANCES OF PUPIL**, all with bounds of **[1:30]**.) The data relevant to each sub-case consists firstly of the major variables in the main case (such as **TEACHERS AGE** and **QUALIFICATIONS**), and secondly, of all the elements in the vectors which have the same suffix (such as **AGE OF PUPIL [3]**, **SEX OF PUPIL [3]**, and so on).

To specify a table in which each sub-case is treated as a complete case, we place, in the table heading, the statement

```
subcase table [m:n];
```

where **m** and **n** are the. bounds of the vectors characterising the subcases. Elsewhere in the specification, we may now refer to items in two different modes; either as single-valued quantities (such as individual items or suffixed vector names) or as unsuffixed vector names. The effect of this is that the case is used for incrementing the table not once but **(n-m+1)** times; that is, once for each possible sub-case. For the first time, all the unsuffixed vectors (except the arguments of vector functions) are given the suffix **m**; for the second time the suffix **[m+1]**; for the third time **[m+2]**, and so on. For the last tabulation, the suffix is **[n]**.

For any one case, the number of increments actually made to the table is the same as the actual number of sub-cases; for the high values of the suffices which correspond to non-existent subcases will yield unknown values and cause those 'pseudocases' to be rejected, by virtue of the rule described in section 2.11.

As the reader will understand, the scope of this facility is so general that it cannot be fully discussed in a few pages. We shall content ourselves with a few trivial examples and leave the reader to discover other applications of the system for himself.

1) To tabulate the pupils' prospects according to the sex of

each pupil In this example, note the use of unsuffixed vector names.

```
title 'PROSPECTS BY SEX OF PUPIL';
sub case table [1:30];
tabulate by SEX OF PUPIL;
count CHANCES OF PUPIL;
finish;
```

This will produce a table similar to:-

```
          PROSPECTS BY SEX OF PUPIL
           GIRL     BOY     TOTAL
TOTAL      8600     7545    16145
CERTAIN    4318     4530     8848
DOUBTFUL   2113    -1907     4020
BAD        2169     1108     3377
```

2) To tabulate the prospects of each pupil according to the sex of the teacher. In this example, note the use of one of the 'major' variables.

```
title 'PROSPECTS BY SEX OF TEACHER';
sub case table [1:30];
tabulate by TEACHERS SEX
count CHANCES OF PUPIL;
finish;
```

```
          PROSPECTS BY SEX OF TEACHER
           MALE     FEMALE     TOTAL
TOTAL     11457       4688     16145
CERTAIN    7312       1536      8848
DOUBTFUL   3127        893      4020
BAD        1018       2259      3377
```

3) To tabulate the 'term's work' selecting only girls taught

by women. Note the use of a simple variable and an unsuffixed vector in the same expression (in the entry criterion).

```
title 'TERMS WORK OF GIRLS TAUGHT BY WOMEN';
sub case table [1:3C];
include FEMALE and GIRL;
count WORK OF PUPIL;
finish;
```

```
          TERMS WORK OF GIRLS TAUGHT BY WOMEN
FEMALE and GIRL.
TOTAL         3412
OUTSTANDING     98
GOOD           987
AVERAGE       1816
POOR           511
```

4) To tabulate the prospects of all pupils according to the prospects of the oldest pupil.

In this tabulation the vector suffix is used in referring to the oldest pupil is **[which max(AGE OF PUPIL)]**. Accordingly since this is not an integer, we must replace the polylog question name referring to the prospects of the oldest pupil by its equivalent list of answer expressions.

```
title 'PROSPECTS BY PROSPECTS OF OLDEST PUPIL';
subcase table [1:30];
tabulate by  CERTAIN [which max (AGE OF PUPIL)],
             DOUBTFUL [which max (AGE OF PUPIL)],
             BAD [which max (AGE OF PUPIL)];
count CHANCES OF PUPIL;
finish;
```

We cannot quote the entire table produced by this specification because it would be too wide for the page, but we give a portion of it below:-

PROSPECTS BY PROSPECTS OF OLDEST PUPIL

	CERTAIN [which max (AGE OF PUPIL)]	TOTAL
TOTAL	7235	16145
CERTAIN	5612	8848
DOUBTFUL	2301	4020
BAD	2113	3377

Finally, let us repeat that great care should be exercised in using this facility, for it is only too easy to specify tables in which the figures are meaningless.

2.13 Multi-dimensional table splits.

The table split facility described in Part I of this manual forms a simple way of defining and tabulating several mutually exclusive classes, which correspond to the possible values of one variable. A simple table split can be thought of as adding a second dimension to a table. Sometimes, it is convenient to make a tabulation for a set of classes which correspond to all possible combinations of values of two or more variables. For example, a survey may contain the two binary variables **EMPLOYED** and **MARRIED**; and it may be necessary to make tabulations for each of the 2 * 2 = 4 classes;

```
EMPLOYED and MARRIED
not EMPLOYED and MARRIED
EMPLOYED and not MARRIED
not EMPLOYED and not MARRIED
```

In MVC, this can easily be done by specifying a multi-dimensional table split. This takes the form

```
tabulate by L1 by L2 by L3 ...  Ln;
```

where **L1**, **L2**, etc. are each lists of mutually exclusive Boolean expressions or the names of polylog questions. Thus, the four groups above could be specified by

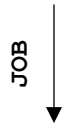
```
tabulate by EMPLOYED, not EMPLOYED by MARRIED, not MARRIED.
```

Each element in the general table split above effectively adds a new dimension to the resulting table. The restricted table split described in Part I is a special case of this more general facility. Consider, for instance, a survey which contained the following polylog variables:-

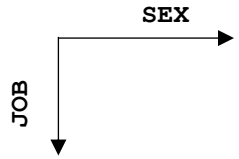
Variable	Possible answers
CLASS	UPPER, MIDDLE, LOWER
SEX	MALE, FEMALE
JOB	MANUAL, OTHER, NONE;

The following table specifications would give rise to tables of 1, 2 and 3 dimensions, respectively.

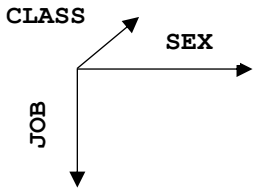
```
title 'ONE DIMENSION';
count JOB;
finish;
```




```
title 'TWO DIMENSIONS';
tabulate by SEX;
count JOB;
finish;
```




```
title 'THREE DIMENSIONS';
tabulate by SEX by CLASS;
count JOB;
finish;
```



When the tables resulting from ,a split with more than one component are printed the sub-tables resulting from permitting the first component of the split are arranged side by side, and the groups of such tables which correspond to combinations of the second and subsequent components are arranged below one another. Thus, a multi-dimensional table is cut into two dimensional 'slices' along planes 'parallel' to the table body and the first component of the table split.

MVC does not impose a limit on the number of dimensions specified in a table split Nevertheless, it should be borne in mind that the number of cells in a table increases rapidly with the number of dimensions, and that since the total number of cases is constant, the average number of cases failing in each cell decreases equally rapidly. Thus, a long table split will result in a very large, set of tables in which most of the cells are empty.

A table heading may include any number of multi-dimensional, as well as single-dimensional table splits.

To illustrate this facility we shall give a possible output produced by the third table specification above.

THREE DIMENSIONS

UPPER

	MALE	FEMALE	TOTAL
TOTAL	165	142	307
MANUAL	5	2	7
OTHER	150	50	200
NONE	10	90	130

MIDDLE

	MALE	FEMALE	TOTAL
TOTAL	670	591	1261
MANUAL	289	173	462
OTHER	385	231	566
NONE	46	187	233

LOWER

	MALE	FEMALE	TOTAL
TOTAL	1045	1123	2168
MANUAL	670	460	1130
OTHER	340	502	842
NONE	35	161	196

2.14 The χ^2 Facility

One of the more advanced facilities provided by the MVC system is that of investigating the association between two qualitative variables by means of the χ^2 test. It must be stressed that this is a very powerful test, which is liable to give misleading results unless carefully and expertly handled. We do not, therefore, recommend its indiscriminate use on all types of data.

When the χ^2 test is specified, the computer forms within itself the appropriate contingency table. It does not normally print this table, but uses.it to compute a value of the statistic χ^2 and the probability (p) that this value could have been exceeded by chance if the variables being tested had no association. Finally, the computer prints the computed value of χ^2 and also an indication of p in the following form:-

$0.05 < p$	nothing
$0.01 < p \leq .05$	*
$0.001 < p \leq 0.01$	**
$p \leq 0.001$	***

The significance indicator normally appears below the coresponding value of X^2.

Yates' correlation is always applied when computing χ^2. If, in any table, the expected value should fall below 2, then the value of χ^2 for that table is suppressed, and a ? is printed instead. If the expected value falls between 5 and 2 then the printed value of χ^2 accompanied by a ? in brackets.

A single χ^2 test, or a matrix of χ^2 tests, may be specified by a statement in a format closely analogous to that used for specifying correlations (see section 2.10.2). The statement begins with the system word

chisquare

which is followed by a list of at least two **polylog** question names, separated by commas, and terminated by a semicolon This format will normally lead to a triangular matrix which gives the association between every possible pair of polylogs. If the list of polylogs is separated into two parts by a **by**, then the matrix will be rectangular.

Whenever two variables show a high degree of association, it will often be useful to obtain the actual contingency table itself, so that the type of association can be examined. The printing of such tables can be specified by including 1,2, or 3 asterisks in the chisquare statement, as in the example below. These will cause printing of any table for which $p \leq 0.05$, 0.01. and 0.001 respectively

Example 1 shows how a single test of association is specified:—

```
chisquare SEX OF TEACHER, QUAL;
```

this would give

```
chisquare  SEX OF TEACHER
QUAL                16.22
                       **
```

Example 2 shows the specification of a rectangular matrix:-

```
chisquare FAMILY BACKGROUND, SEX, RACE by
ACADEMIC SUCCESS SPORTS RESULTS, BEHAVIOUR;
```

```
chisquare  FAMILY BACKGROUND    SEX    RACE
ACADEMIC SUCCESS       13.37   7.24    7.91
                           *              (?)
SPORTS RESULTS         22.70  14.00   35.89
                          **     **     ***
BEHAVIOUR                  ?   8.99   12.57
                                          *
```

Example 3 refers to a triangular matrix and contains a request to print any tables in which the variables have significant ($p < 0.001$) association.

```
chisquare SEX, AGE GROUP, MARITAL STATUS, POLITICS, ***;
```

```
CHISQUARE          SEX   AGE GROUP  MARITAL STATUS

AGE GROUP         3.24

MARITAL STATUS    8.17      645.11
                     *         ***
POLITICS         11.91       47.19            12.30
                                **                *
```

MARITAL STATUS/AGE GROUP

	YOUNG	MIDDLEAGED	OLD	TOTAL
SINGLE	497	314	219	1030
MARRIED	518	837	316	1471
WIDOWED	4	65	185	254
DIVORCED	25	157	42	224
TOTAL.	844	1375	762	2979

When using the test facility, several important points should be borne in mind:-

1) The chisquare statement may only refer to polylog question names. If it is desired to form 'two by two' tables referring to binary variables, then it is necessary to define corresponding derived polylogs, in a suitable manner; for example:-

```
derived polylog;
TYPE OF CASE := TOOTHACHE, not TOOTHACHE: EXPERIMENTAL, CONTROL;
```

(Here **TOOTHACHE** is supposed to be a binary variable.)

2) The χ^2 test may sometimes yield indeterminate results because the polylogs being compared have too many possible values, with too few cases in each. If this is thought likely, it is possible to overcome this difficulty by defining derived polylogs in which the smaller classes are 'lumped' together. For example:-

```
LUMPED RELIGIONS := (PROTESANT or CATHOLIC), (JEWISH or
MUSLIM or HINDU or BHUDDIST), (AGNOSTIC or ATHEIST):
CHRISTIAN, OTHER FAITH, NO FAITH;
```

3) Due to the idiosyncracies of the Atlas operating system, it is. necessary to separate consecutive asterisks in chisquare statements by spaces. THIS IS VERY IMPORTANT..

Chisquare statements may be included in tables of which the heads contain weights, table splits, entry criterions, and so on.

2.15 MVC PHASES

Provisional Description

This section describes the operating procedures to be followed in using the MVC system in its present state. As it nears completion, various changes to the operating system will be announced in the MVC bulletin.

The section assumes that readers are familiar with JOB DESCRIPTION and with the general operating system on ATLAS. Those who are not acquainted with the system should read no further, but apply to the author for advice.

The analysis of any survey by the MVC system takes place in three distinct 'phases', which must normally follow one another in time. The only link between the phases is a private magnetic tape, which must be obtained by the user.

During Phase I, the user's survey specification (but not his data) is read and checked for errors of form such as ambiguous definitions or mis-spelt system words. If found correct, the specification is stored in a translated form on the user's private magnetic tape. If any errors are found, the specification is not stored, but a description of the errors is printed.

(At the time of writing, when the system is still under development, Phase I also prints a complete listing of the translated specification. This listing is of no direct interest to the user, but should be kept to show the author if the subsequent survey does not proceed as expected.)

In Phase II, the user's data is read. For every case supplied, all the variables are computed according to the specification, and any checks which have been requested are carried out. The case number of each case presented is printed. When Phase II is used in the 'normal' mode, then the variables which represent each satisfactory case are stored on the private tape; the values of the variables of each cases which fail are printed in full.

When used in the 'checking' mode, Phase II effectively rejects every case, printing the variables for each one presented, and not storing any on the private tape. The checking mode has been provided so on that Phase II can be used with a few 'sample' cases to ensure that the content, as well as the form, of the specification is correct. A careful examination of the listing produced by this phase, and a comparison of the values printed against those expected for the sample cases, will quickly show up such errors as incorrect column numbers, wrong answer formats, or incorrect case number declarations.

All users are strongly urged to use this type of checking on their specification. A mistake discovered at this stage may well prevent the reading of a very large amount of data to no useful purpose.

Phase III is used to generate and print the actual tables asked for in the specification. The phase takes the variables it needs from the private magnetic tape.

Although an 'ideal' survey would be analysed by the simple sequence of phases I, II and III there are great practical advantages in allowing the phases to be used individually in a flexible manner. For example, phase II should always be used in the checking mode before the normal mode. Again, if the amount of data is large, it is advisable to read in several batches, each one corresponding to a separate run of phase II.

Yet again, it may be necessary to follow up an initial 'exploratory' analysis by more detailed tables of sections of the data found to be interesting. This can be done by a new run of phase I (to read the new table specifications) and a new run of phase III (to generate the tables). Phase II will not be needed again, since all the data will already have been stored on the private tape.

The operation of the phases is controlled by a special program called the 'MVC supervisor'. The first item in any MVC job, after the job description, is a set of instructions, to this supervisor, about what phases to carry out on this particular run. At present, these instructions must be written in a very simple and rigid form as shown below:.

P0	Carry out Phase I
P1	Carry out Phase II in the checking mode.
P2	Carry out Phase_ II in the normal mode
P3	Carry out Phase III
Q	End of instructions to the MVC supervisor.

These instructions can be combined together; for example

P0P1Q

implies that the (given) specification should be translated by Phase I and then phase II should be used to check the (given) sample cases; or

P2P3Q

indicates that the data supplied should be read, stored on tape, and previously defined tables constructed.

When typing instructions to the MVC supervisor, great care should be taken to ensure that the **P** is immediately followed by the relevant decimal digit, without any spaces, erases, backspaces, or changes. THIS IS IMPORTANT.

2.15.1. Job Descriptions and Documents.

At the time of writing, it is very difficult to assess the relationship between the constants required in the job description and the size of any survey. The following rules, however, will provide a rough and ready guide:

a) Storage: 60 blocks

b) Computing time:

For Phase I, 1000 instructions

For Phase II, 50 instructions per case (if on tape) or per card (if cards are used)

For Phase III,10000 instructions. + 10 per case

c) Output: 5000 lines (on output 0)

These figures are excessive for most surveys.

2.15.2. Compiler Specification

At the time of writing, MVC has not yet been defined as a compiler. One cannot therefore put

```
COMPILER MVC
```

Instead, the following ABL sequence is required:

```
COMPILER ABL
1001,0,0,1
1002,0,0,99:0
121,127,0,99:0
EA0/0
```

2.15.3. Specification of Private Tape

At present the job description must contain the sequence

```
TAPE
0 L0005
```

2.15.4. Input Documents

The survey specification is always read on input 0, and should follow on the same tape as the job description.

Data on punched tape is also read on input 0, and should follow the specification (if any). It should be terminated by ***Z. If the data is on a separate tape, then the specification should end with ***T.

Data on punched cards is read on input 1. The cards should be preceded by two title cards, the first punched '**DATA**' in cols. 1-4, and the second with a suitable data title. This punching must be in the ICT code. The data cards should be followed by a single card with holes 7 and 8 punched in column 1 and a '**Z**' in column 80.

When cards are read, the job description must contain the statement

```
INPUT
1 < The data title >
```

2.15.5. Examples of Job Description

1) To read a specification, and to check a few cases, where the data is on punched tape.

```
JOB
LRC92PA1, ROAD SURFACE SURVEY
STORE 60 BLOCKS
COMPUTING 2000 INSTRUCTIONS OUTPUT
0 LINE PRINTER 5000 LINES
TAPE
0 L0005
COMPILER ABL
1001,0,0,1                    ≡ COMPILER MVC
1002,0,0,99:0
121,127,0,99:0
EA0/0
P0P1Q                         INSTRUCTIONS TO MVC SUPERVISOR
read items;                   Survey Specification
..........
start;
.....                         Sample cases
.....
***Z
```

2) To read data on cards, and to tabulate it.

```
JOB
LRC92PA1, TYRE WEAR SURVEY, J. BLOGGS
STORE 60 BLOCKS
COMPUTING 45000 INSTRUCTIONS
OUTPUT
0 LINE PRINTER 5000 LINES
INPUT
1 J. BLOGGS TYRE DATA
TAPE
0 L0005
COMPILER ABL
1001,0,0,1
1002,0,0,99:0
121,127,0,99:0
P2P3Q
***Z
```

The data cards would be headed by the two cards:

```
DATA
J. BLOGGS TYRE DATA
```

Note that the survey specification is absent, and that phase I is not specified. This job could only run if the actual specification had been read and stored on the private tape by a previous run.

2.15.6. Restrictions and Points to Watch

At present, the MVC system will only accept cards on which the individual columns are punched in a way which represents some actual character in the ICT code. This means that with a few special exceptions, double punching is not permissible.

When any of the phases are run for a second time on the same job, the following points should be borne in mind:

Phase I The only purpose for repeating phase I in any job is that of defining new table specifications. When this is done, then the variable definitions in the new specification must be exactly the same as in the original one. Only the table specifications may be changed.

Phase II Phase II can be repeated several times if the total amount of data is large, or if it is desired to re-read cases which were rejected on an earlier pass and have since been corrected. Additional runs of phase II can be started without any formality. The case numbers must be in ascending order on any one run, but need not follow on from the previous one.

Phase III can be repeated as many times as necessary, without formality.

AJTC:JL

15 October 1964

SN:41