JAN AAGAARD

Data Entry System

PLANNING

3600

introduction to the FORMAT LANGUAGE

RC 3600 DATA ENTRY SYSTEM

INTRODUCTION TO THE FORMAT LANGUAGE

The technical information in this document, while correct at the time of publication, is liable to change without notice.

A/S Regnecentralen
Documentation Department

Revision 0
February 1975

CONTENTS

1. INTRODUCTION

Format and subformat

The initial keying of data fields, output of records to disc, rekeying verification, and editing of the batch are all controlled by a specified record format, as is the creation of new formats, subprograms, and tables.

A record format contains a number of subformats, each of which controls the input and output of a specific type of record. The subformats may be entered in random sequence, but a new subformat cannot be entered until the current subformat is finished. New subformats can be selected by the program or by the operator.

Each subformat consists of a number of sequentially entered field descriptions, which together define the structure of the record in question.

An example of a format is given in Section 2.

Field description

Each field description in the subformat has two parts: the so-called checkbox part and a program part with statements in the Format Language.

The checkbox part, which contains a number of standard parameters, is used for various functions including a basic validity check of the input data and definition of the output format. One can also specify in the checkbox part whether a keyed field is to be displayed and whether the field in question must be rekeyed for verification. Finally, the name of a register containing a constant or duplication value can be specified.

An explanation of the checkbox parameters is given in Section 3.

The program part of the field description is used to modify or supplement the standard parameters of the checkbox part. Field programs can operate on all of the keyed fields in the subformat as well as on a number of registers associated with the individual keystations and used for the transfer of data from one record to another.

Other program features include the use of range sets, tables of legal or illegal values, automatic insertion, automatic duplication, automatic skipping, arithmetic operations, display of error and other operator messages, and check digit verification.

The use of many of the Format Language statements is illustrated by the example in Section 2, while a complete survey is given in Section 4.

2. EXAMPLE OF A FORMAT

A format for input of an invoice

Fig. 1 shows a format for the input of an invoice containing three categories of information: (1) customer information, (2) information on the articles shipped to him, and (3) the total price. The format contains, accordingly, three subformats, one for each type of record.

Subformat 1 controls the keying of the customer data (head of the invoice).

Subformat 2 controls the keying of the article data (one line in the body of the invoice for each article).

Subformat 3 controls the keying of the total price, checking it against the totalled up price (end of the invoice).

The keying is controlled by the current subformat until a new subformat is selected. In this format, subformat 1 is automatically selected as the first subformat. When subformat 1 is finished, the program automatically selects subformat 2, which is repeated until there are no more articles to be invoiced. The operator then selects subformat 3, ending the format.

Each of the three categories of information contained in the invoice consists of one or more items of information. Thus the first category consists of the customer's number, discount percentage, name, street address, postal number, and city. Similarly, the second category includes six "article items," while the third category has only one item, namely, the total price.

Each subformat consists, accordingly, of one or more field descriptions, one for each field in the type of record concerned. Thus subformats 1, 2, and 3 consist, respectively, of six, six, and one field description.

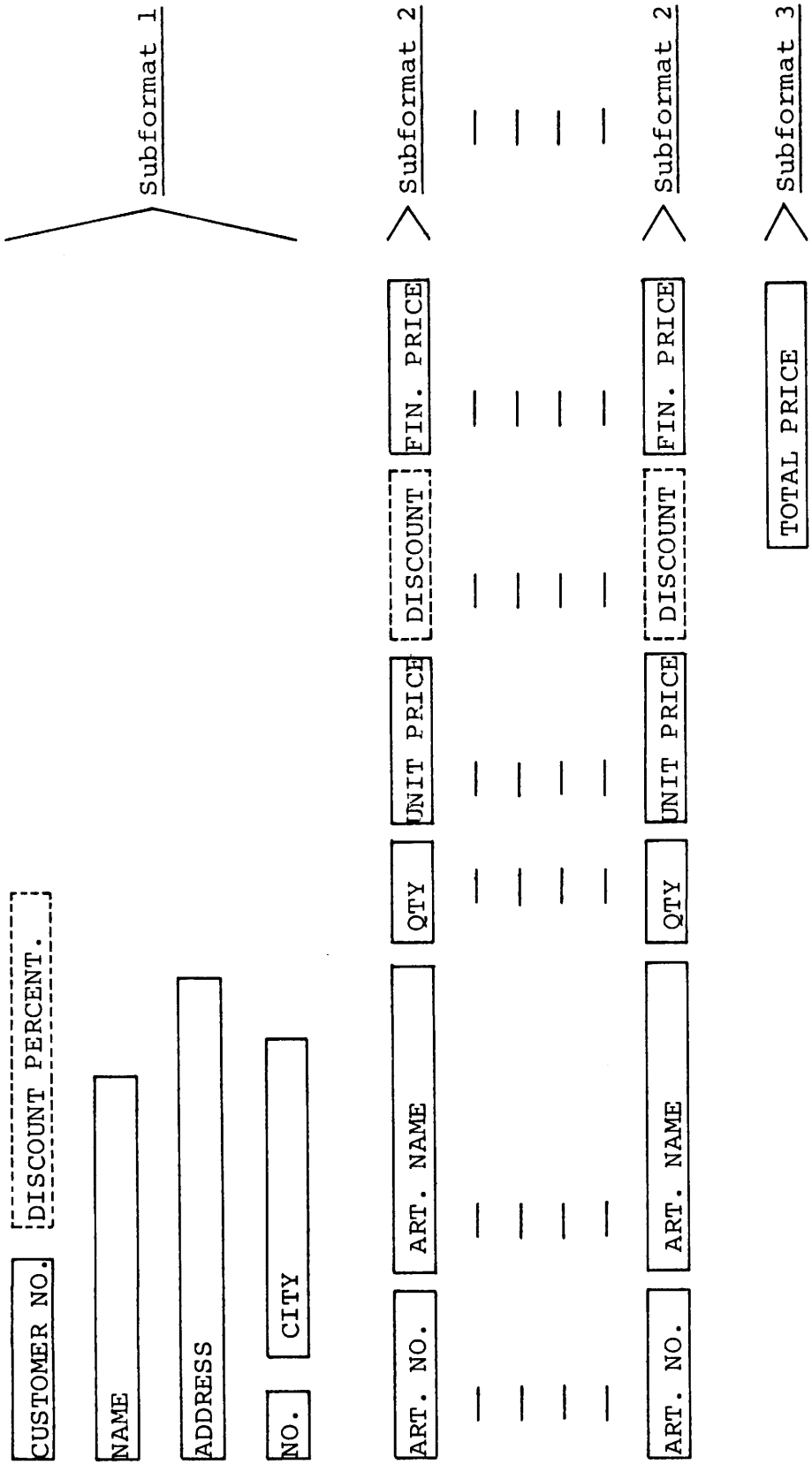
How the format is coded

The exact coding of the three subformats comprising the format program just described is shown on the pages immediately following Fig. 1.

Each of the subformats has a subformat head containing the name of the format, the name of the subformat, and a comment. This is followed by the individual field descriptions, each of which begins with a specification of the checkbox parameters and ends with one or more program statements.

The coding of the subformats will be elucidated by reference to Section 3 (checkbox parameters), Section 4 (program statements), and Section 5, which contains a complete, formal description of the Format Language.

Fig. 1. Format for input of an invoice



FORM NAME	Subform	COMMENT
1	2	3
EXAMP	1	SUBFORMAT 1 - HEAD OF INVOICE

FIELD NAME	Length	Min. length	Type	Output position	R/1	Fill	Verify	Display	Kind	Register	PROGRAM STATEMENTS
1	2	3	4	5	6	7	8	9	10	11	12
CUSNO	12	8	AN	1	L						NOTE READ CUSTOMER NUMBER - SEARCH IN CUSTOMER TABLE FOR DISCOUNT PERCENTAGE AND DUMP IT, DEFINE X01 2, SEARCH CUSNO IN CTABL GIVING X01 AT END ALARM 'CUSTOMER NUMBER NOT KNOWN',
DISCT	2	2	N	2					N		NOTE INSERT DISCOUNT PERCENTAGE IN RECORD, COMPUTE DISCT = X01,
NAME	22	1	A	3	L						NOTE READ NAME,
ADRES	28	1	AN	4	L						NOTE READ ADDRESS,
PCODE	4	4	N	5							NOTE READ POSTAL CODE, LIMIT 1000 9000, NOTE LIMITS 1000 <= FIELD <= 9000,
CITY	18	1	AN	6	L						NOTE READ CITY,
	0										NOTE DUMMY FIELD FOR RESETTING REGISTERS, DEFINE X03 10, NOTE FINAL PRICE, DEFINE X04 15, NOTE TOTAL PRICE, COMPUTE X04 = 0, DEFINE X05 10, NOTE REGISTER FOR SEARCHING ARTICLE, SELECT SUBFORMAT 2, END SUBFORMAT,

FORM NAME	Subform	COMMENT
1	2	3
EXAMP	2	SUBFORMAT 2 - ONE LINE IN BODY OF INVOICE

FIELD NAME	Length	Min. length	Type	Output position	R/1	Fill	Verify	Display	Kind	Register	PROGRAM STATEMENTS
1	2	3	4	5	6	7	8	9	10	11	12
ARTNO	10	10	N	1							NOTE READ ARTICLE NUMBER AND PERFORM CHECK DIGIT VERIFICATION (NO 10), PERFORM CHE10, IF ARTNO = 0 THEN SKIP 1 FIELD,
NAME	18	1	AN	2	L						NOTE READ ARTICLE NUMBER - SEARCH IN ARTICLE TABLE FOR CONNECTION BETWEEN NUMBER AND NAME, SEARCH NAME IN ATABL GIVING X05 AT END ALARM 'ARTICLE NAME NOT KNOWN', IF X05 <> ARTNO THEN ALARM 'ARTICLE NUMBER AND NAME DO NOT MATCH',
QTY	6	1	N	3	R	0					NOTE READ QUANTITY,
UNITP	10	1	N	4	R	0					NOTE READ UNIT PRICE - COMPUTE PRICE AND DUMP IT, COMPUTE X03 = QTY * UNITP,
DISCT	10	10	N	5					N		NOTE INSERT DISCOUNT, COMPUTE DISCT = (X03 * X01) / 100,

FINAL	10	1	N	6	R	0					NOTE READ FINAL PRICE AND CHECK AGAINST COMPUTED PRICE,
											IF FINAL <> X03 - DISCT THEN
											ALARM 'FINAL PRICE NOT OK'
											ELSE
											NOTE UPDATE TOTAL PRICE;
											COMPUTE X04 = X04 + X03 - DISCT,
											END SUBFORMAT,

FORM NAME	Subform	COMMENT
1	2	3
EXAMP	3	SUBFORMAT 3 - END OF INVOICE

FIELD NAME	Length	Min. length	Type	Output position	R/L	Fill	Verify	Display	Kind	Register	PROGRAM STATEMENTS
1	2	3	4	5	6	7	8	9	10	11	12

TOTAL	15	1	N	1	R	0					NOTE READ TOTAL PRICE AND CHECK IT AGAINST TOTALLED UP PRICE,
											IF TOTAL <> X04 THEN
											ALARM 'TOTAL PRICE NOT OK'
											ELSE
											DISPLAY 'END OK',
											END,

3. CHECKBOX PARAMETERS

Parameters

A field description begins with the name of the field (1 to 5 characters) and ends with one or more statements in the Format Language. Between the FIELD NAME and the PROGRAM STATEMENTS, the following ten parameters are specified:

Length	The length of the output field and the maximum length of the input field: 0 to 80 characters.
Min. length	The minimum length of the input field: 0 to 80 characters.
Type	The type of characters of which the input field should consist: numeric (N), signed numeric (SN), alphabetic, including punctuation (A), or alphanumeric (AN).
Output position	The position of the field in the output record: field number from 1 up. (Dummy fields, where "length" is equal to zero, are excluded). This parameter can be used for reformatting in a form more compatible with computer processing.
R/l	When the number of keyed characters is less than "length," the justification of the characters in the output field: right (R) or left (L).
Fill	When the number of keyed characters is less than "length," the fill character to be used in the output field: zero (0), asterisk (*), or space ( ).
Verify	Verification of the field in rekey mode: verify ( ) or do not verify (N).
Display	Display of a keyed field on the keystation display screen: display (Y) or do not display (N).

Kind	The kind of field input required: keyed ( ), not keyed (N), constant (C), semi-constant (SC), duplication (D), or semi-duplication (SD).
Register	A register containing a constant or duplication value: register name (<digit><digit>).

Normal, empty, and continue checkboxes

In a normal checkbox, "length" is greater than zero. The parameters "min. length," "type," and "output position" may not have default values (see below).

In an empty checkbox (dummy field), "length" is equal to zero. All of the other parameters must be blank (i.e. have default values). An empty checkbox requires no keying, and the field program is always executed.

In a continue checkbox, "length" is blank (i.e. has a default value). All of the other parameters must also be blank. A continue checkbox is used to continue the program part of the last normal or empty checkbox.

Default values

FIELD NAME	'ΔΔΔΔΔ'	(*)
Length	'ΔΔ'	*
Min. length	'ΔΔ'	*
Type	'ΔΔ'	*
Output position	'ΔΔ'	*
R/l	R	
Fill	'Δ'	
Verify	'Δ'	
Display	'X'	
Kind	'Δ'	
Register	'ΔΔ'	

Δ - blank.

* - must be specified, if "length" is greater than zero.

Kinds of field input

When a keyed field is specified, normal field input is required, i.e. the operator should key the field.

When a not keyed field is specified, field input is impossible, i.e. the operator should not do anything, as only the program statements are executed.

When a constant, semi-constant, duplication, or semi-duplication field is specified, the operator should press the DUP key, so that the constant in the specified register is used as field input; otherwise normal field input is required. This is explained below.

Automatic insertion

Automatic insertion of a constant in a field can be performed in several ways.

Always insertion

1. Field specification: not keyed field (N)

   Program statement: COMPUTE field = 'constant'
   MOVE 'constant' TO field

   Operator action: None.

   Effect: The program statements are executed.

2. Field specification: constant field (C)
   register name (<digit><digit>)
   The specified register is initialized in a previous subformat.

   Operator action: Press the DUP key.

   Effect: The constant in the specified register is used as field input.

Semi-automatic insertion

3. Field specification:		semi-constant field (SC) register name (<digit> <digit>) min. length > 0
		The specified register is initialized in a previous subformat.
	Operator action:	Press the DUP key.
	Effect:	The constant in the specified register is used as field input.
	Operator action:	Key <data>. Press the ENTER key.
	Effect:	As for normal data entry input.

Automatic duplication

Automatic duplication of a field in the corresponding field in the next record can be performed as follows:

Always duplication

1. Field specification:		duplication field (D) register name (<digit> <digit>)
	Operator action:	Key <data>. Press the ENTER key.
	Effect:	In the first record, the input value is placed in the specified register. Not accepted in any other record.
	Operator action:	Press the DUP key.
	Effect:	Not accepted in the first record. In any other record, the constant in the specified register is used as field input.

Semi-automatic duplication

2. Field specification:		semi-duplication field (SD)
	Operator action:	Key <data>. Press the ENTER key.
	Effect:	The input value is placed in the specified register.
	Operator action:	Press the DUP key.
	Effect:	Not accepted in the first record. In any other record, the constant in the specified register is used as field input.

Automatic skipping

Automatic skipping of a field can be performed as follows:

Always skipping

1. Field specification:		not keyed field (N)
	Program statement:	SKIP 'constant' FIELDS
	Operator action:	None.
	Effect:	The program statement is executed.

Semi-automatic skipping

2. Field specification:		keyed field ( ) min. length = 0
	Operator action:	Key <data>. Press the ENTER key.
	Effect:	As for normal data entry input.
	Operator action:	Press the ENTER key.
	Effect:	The field is skipped.

4. FORMAT LANGUAGE STATEMENTS

Assignment statement

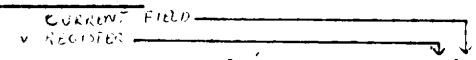
Syntax: COMPUTE <variable> = <expression>

The destination variable must be the current field or a register. The current field is allowed as the destination variable only if it is a not keyed field. The expression is evaluated and the result is stored in the destination variable.

Example: COMPUTE DISCT = (X03 * X01) / 100,

where X03 = 2000 and X01 = 3 causes DISCT = 60.

Move statement



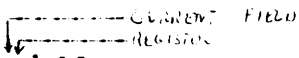
Syntax: MOVE <variable> TO <variable>

The destination variable must be the current field or a register. The current field is allowed as the destination variable only if it is a not keyed field. The variables must be alphanumeric in type. The first variable is moved to the second variable.

Example: MOVE 'TEXT' TO X02,

causes X02 = 'TEXT'.

Connect statement



Syntax: CONNECT <variable> TO <variable> GIVING <variable>

The destination variable must be the current field or a register. The current field is allowed as the destination variable only if it is a not keyed field. The two source variables and the destination variable must be alphanumeric in type. The two source variables are catenated and the result is stored in the destination variable.

Example: CONNECT A TO B GIVING X01,

where A = 'ABC' and B = 'DEF' causes X01 = 'ABCDEF'.

Search statement

Syntax: SEARCH <variable> IN <table identifier> GIVING <variable>
AT END <unconditional statement>

This statement is used to search in a table for a specified argument. If the search is successful, the function of the argument is stored. If the search is unsuccessful, the unconditional statement is performed.

The table must be double-entried. The source variable must be of the same type as the table arguments, and the destination variable must be equivalent to the table functions. The destination variable must be the current field or a register. The current field is allowed as the destination variable only if it is a not keyed field.

```
Example: SEARCH CUSNO IN CTABL GIVING X01 AT END
               ALARM 'CUSTOMER NUMBER NOT KNOWN',
```

Limit statement

Syntax: LIMIT <number><number>

This statement is used to check the value of the current field against two limits. The current field must be numeric in type and a normal keyed field. If the check fails, the program is interrupted, and the operator must either correct or bypass the field.

```
Example: LIMIT 1000 9000,  NOTE  LIMITS 1000 <= FIELD <= 9000,
```

Allow and disallow statements

Syntax: ALLOW <list> | ALLOW <table identifier>

Syntax: DISALLOW <list> | DISALLOW <table identifier>

These statements are used to check the current field for specific values. Tables may be single-entried or double-entried. The current field, which must be alphanumeric in type, is checked against the table arguments, which must also be alphanumeric. Errors are treated as for the limit statement.

```
Example: ALLOW 'HANSEN' 'JENSEN',
```

```
Example: ALLOW CTABL,  NOTE  CTABL IS A TABLE,
```

Goto statement

Syntax: GOTO <label>

The label must be defined in the same field program. The execution of the current field program is continued from the specified label.

Example: GOTO STOP,
-
-
STOP: DISPLAY 'END',

Select statement

Syntax: SELECT SUBFORMAT <subformat>

This statement may appear only in the last field program in a subformat. The current subformat (record) is terminated and the specified subformat is entered.

Example: SELECT SUBFORMAT 2,

Display statement

Syntax: DISPLAY <string>

This statement displays the string on the second line (message part) of the keystation display screen. It can be used for fill-in-the-blanks keying.

Example: DISPLAY 'END OK',

Alarm statement

Syntax: ALARM <string>

This statement displays the string on the second line (message part) of the keystation display screen. The field must then be corrected or bypassed by the operator.

Example: ALARM 'FINAL PRICE NOT OK',

Define statement

Syntax: DEFINE <register> <unsigned number>

This statement is used to define the length of a register in characters. The define statement may only be used before a register is used for the first time, and all registers must be defined before they are used. The next define statement pertaining to the same register may not change the length.

Example: DEFINE X03 10,

End statement and end subformat statement

Syntax: END

Syntax: END SUBFORMAT

The end statement terminates the whole format program. It must occur in the last field in the last subformat. The end subformat statement terminates the current subformat. It must occur as the last statement in the last field in the current subformat.

Example: END,

Example: END SUBFORMAT,

Note statement

Syntax: NOTE <characters>

This statement is used for comments. If quotation marks are used, they must occur in pairs. As the note statement is terminated by a comma or a semicolon, it may not be used as the last statement before ELSE (because <sentence> is not terminated by a comma or a semicolon; see below under "Conditional statement").

Example: NOTE READ NAME,

Procedure statement

Syntax: PERFORM <procedure identifier>

The procedure identifier must be the name of a translated subprogram (see further in Section 5 under "Statements in procedures").

Example: PERFORM CHE10,

Skip statement

Syntax: SKIP <unsigned number> FIELDS

The unsigned number must be greater than 0 and less than 256. The execution of the format is continued <unsigned number> of fields after the current field.

Example: SKIP 1 FIELD,

Conditional statement

Syntax: IF <expression> THEN <sentence> |

IF <expression> THEN <sentence> ELSE <sentence>

The boolean expression following IF is evaluated. If the condition is true, the sentence following THEN is executed. If the condition is false, the sentence following THEN is skipped and the sentence following ELSE is executed.

```
Example: IF TOTAL <> X04 THEN
               ALARM 'TOTAL PRICE NOT OK'
         ELSE
               DISPLAY 'END OK',
```

5. FORMAT LANGUAGE SYNTAX

Character set

```
<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
```

<space>	::=		<semicolon>	::=	;
<plus>	::=	+	<quotation>	::=	'
<minus>	::=	-	<left>	::=	(
<asterisk>	::=	*	<right>	::=	)
<stroke>	::=	/	<greater than>	::=	>
<equal>	::=	=	<less than>	::=	<
<comma>	::=	,	<colon>	::=	:

Operands

```
<alphanum>      ::= <letter> | <digit>
<identifier>    ::= <letter> | <letter> { <alphanum> }  (4, 0)
```

Reserved names (see below) may not be used as identifiers.

```
<register>      ::= X01 | X02 | .... | X99
<field>         ::= <identifier>
<variable>      ::= <field> | <register> | <field> (<unsigned
                    number>) | <register> (<unsigned number>)
```

Fields are numeric or alphanumeric depending on their type as specified in the checkbox part of the field description. A destination operand must be a register or a current field. The current field is allowed as a destination, only if it is of the not keyed kind (i.e. N). Any fields before the current field in the current record may be used as sources. The current field is allowed as a source, only if it is not of the not keyed kind (i.e. K, C, or D).

Registers are numeric or alphanumeric depending on their contents. Subscripts are used to refer to individual characters within a register or a field. A subscript is an unsigned number. Subscripts should be in the range 1 - as the leftmost subscript - to the length of the register or the field.

Constants can be either unsigned numbers or character strings. Strings are always enclosed in quotation marks, and must not be interrupted by a continue checkbox.

<sign> ::= - | +

<unsigned number> ::= <digit> | <digit><unsigned number>

<number> ::= <sign><unsigned number> | <unsigned number>

<string> ::= '{<character>}$_{0}^{78}$'

<constant> ::= <string> | <unsigned number>

<label> ::= <identifier>

Labels must be defined within the field program that contains the reference to the label, and may not refer to labels in other field programs.

<subformat> ::= <alphanum>

<list> ::= <string> | <string><list>

<procedure identifier> ::= <identifier>

A procedure is a subprogram that is called by a format program.

<table identifier> ::= <identifier>

A table is either single-entried or double-entried. All arguments in a table must be of the same length and type. All functions in a double-entried table must be of the same type.

Operators

<relational operator> ::= > | < | = | >= | <=

> means greater than, < less than, = equal to, >= greater than or equal to, and <= less than or equal to.

<table operator> ::= IN

IN is an operator for searching in a table.

<adding operator> ::= + | - | OR

+ means addition, - subtraction, and OR logical 'or'.

<multiplying operator> ::= * | / | AND

* means multiplication, / division, and AND logical 'and'.

<negating operator> ::= NOT

NOT means logical 'not'.

Expressions

```
<expression>          ::= <simple expression> |
                          <simple expression><relational operator><simple expression> |
                          <simple expression><table operator><table identifier>
<simple expression>   ::= <term> | <adding operator><term> |
                          <simple expression><adding operator><term>
<term>                ::= <factor> | <term><multiplying operator><factor>
<factor>              ::= <constant> | <variable> | (<expression>) |
                          <negating operator><factor>
```

An expression is evaluated with respect to the following precedence of operators: (1) negating operator, (2) multiplying operator, (3) adding operator, and (4) relational operator / table operator. Sequences of operators having the same precedence are executed from left to right.

Examples

```
Factors:              15
                      X01
                      CUSNO
                      X05 (10)
                      (X03 * X01)
                      NOT ((A > B) AND (CUSNO IN CTABL))

Terms:                X02
                      X03 * X01
                      X03 * X01 / 100
                      (A > B) AND ((X03 * 2) > 10)

Simple expressions:   X03
                      - CUSNO
                      A * B + C * D

Expressions:          X04
                      X05 = 'TEXT'
                      X06 IN TAB
                      (A + B) > 3
```

Types for operators

Internally, the types of fields are numeric, if the type specified in the checkbox part of the field description is numeric (N) or signed numeric (SN), and alphanumeric, if the type specified is alphabetic (A) or alphanumeric (AN).

Registers are numeric or alphanumeric, depending on their contents.

Constants are numeric, if the item is <number> or <signed number>, and alphanumeric, if the item is <string>.

Table elements are numeric, if the type specified in the table head is numeric (N), and alphanumeric, if the type specified in the table head is alphanumeric (AN).

Relational operators require operands of the numeric or alphanumeric type. The operands must be identical in type. The result of a relation is boolean in type.

The table operator expects the first operand to be a simple one, i.e. a variable or a constant, and the second operand to be a table, either single-entried or double-entried. The type of the first operand must match the type of the arguments in the table. The type of the result is boolean.

Adding operators

Operator	Operation	Type of operands	Type of result
+	addition	numeric[1]	numeric
-	subtraction	numeric[1]	numeric
OR	logical 'or'	boolean	boolean

[1] When used as operators with only one operand, + denotes the identity operation and - denotes sign inversion.

Multiplying operators

Operator	Operation	Type of operands	Type of result
*	multiplication	numeric	numeric
/	division	numeric	numeric
AND	logical 'and'	boolean	boolean

The negating operator requires one operand of the boolean type. The negation is of the boolean type.

Reserved names

ALARM	MOVE
ALLOW	NOT
AND	NOTE
AT	OR
COMPUTE	PERFORM
CONNECT	SEARCH
DEFINE	SELECT
DISALLOW	SKIP
DISPLAY	SUBFORMAT
ELSE	THEN
END	TO
FIELD	X00
GIVING	X01
GOTO	X02
IF	-
IN	-
LIMIT	X99

Only the five leading characters are significant. Thus PERFORM is equivalent to PERFO, for example, and PERFO is equivalent to PERFORMANCE, but IN is not equivalent to INCORRECT.

Statements

Unconditional statements

```
<unconditional statement>   ::=  <assignment statement> | <move statement> |
                                 <connect statement> | <search statement> |
                                 <limit statement> | <allow statement> |
                                 <disallow statement> | <goto statement> |
                                 <select statement> | <display statement> |
                                 <alarm statement> | <define statement> |
                                 <end statement> | <end subformat statement> |
                                 <note statement> | <procedure statement> |
                                 <skip statement>
<assignment statement>      ::=  COMPUTE <variable> = <expression>
```

```
<move statement>            ::= MOVE <variable> TO <variable>
<connect statement>         ::= CONNECT <variable> TO <variable>
<search statement>          ::= SEARCH <variable> IN <table identifier>
                                GIVING <variable> AT END
                                <unconditional statement>
<limit statement>           ::= LIMIT <number><number>
<allow statement>           ::= ALLOW <list> | ALLOW <table identifier>
<disallow statement>        ::= DISALLOW <list> | DISALLOW <table identfier>
<goto statement>            ::= GOTO <label>
<select statement>          ::= SELECT SUBFORMAT <subformat>
<display statement>         ::= DISPLAY <string>
<alarm statement>           ::= ALARM <string>
<define statement>          ::= DEFINE <register><unsigned number>
<end statement>             ::= END
<end subformat statement>   ::= END SUBFORMAT
<note statement>            ::= NOTE <string>
<procedure statement>       ::= PERFORM <procedure identifier>
<skip statement>            ::= SKIP <unsigned number> FIELDS
```

Conditional statement

```
<conditional statement>     ::= IF <expression> THEN <sentence> |
                                IF <expression> THEN <sentence>
                                ELSE <sentence>
<sentence>                  ::= <label> : <sentence> | <unconditional statement> |
                                <unconditional statement> ; <sentence>
```

Program

```
<program>                   ::= <empty> | <statement> | <program><statement>
<statement>                 ::= <label> : <statement> | <unconditional statement>,
                                <conditional statement>,
```

Semantics

See Section 4.

Statements in procedures

All of the statements except the select statement and the end subformat statement may be used in procedures.

Only registers are allowed as variables in procedures.

A procedure must be terminated by an end statement.

An explanation of the notation used

The following examples illustrate the principles on which the notation used to describe the Format Language is based.

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

This means that a "digit" is defined as 0
or 1 or 2 or 3 or 4 or 5 or 6 or 7 or 8 or 9.

<variable> ::= <field> | <register> |
<field> (<unsigned number>) |
<register> (<unsigned number>)

This means that a "variable" is defined as a "field" or a "register"
or a "field" followed by an "unsigned number" enclosed in parentheses
or a "register" followed by an "unsigned number" enclosed in parentheses.

<unsigned number> ::= <digit> | <digit><unsigned number>

This means that an "unsigned number" is defined as a "digit"
or a "digit" followed by an "unsigned number" (i.e. a number of "digits").

<string> ::= '{<character>}$_{0}^{78}$'

This means that a "string" is defined as
0 to 78 "characters" enclosed in quotation marks.

<skip statement> ::= SKIP <unsigned number> FIELDS

This means that a "skip statement" is defined as
SKIP followed by an "unsigned number" followed by FIELDS.

<sentence> ::= <label> : <sentence> |
<unconditional statement> |
<unconditional statement> ; <sentence>

This means that a "sentence" is defined as
a "label" followed by a colon followed by a "sentence"
or an "unconditional statement"
or an "unconditional statement" followed by a semicolon
followed by a "sentence".