CPB-147

GE 225

GENERAL ELECTRIC

TABSOL MANUAL

(Preliminary)

GE 225

This document is a draft of a Preliminary Reference Manual and a language specification for integrating decision tables with the General Compiler. The information contained herein assumes a basic knowledge of computers and electronic data processing applications. Therefore, the manual should be used not as a text book but rather to augment already realized skills. Minor changes in language specifications may occur during the implementation period of the compiler. Any changes that are made will be reflected in future and more final versions of this manual or in supporting material issued during the interim of implementation.

I INTRODUCTION

Early automating coding systems, such as assembly programs, employed mnemonic abbreviations in place of the computer's numerical instruction code and symbolic addresses in place of actual memory addresses. In reality, the assembly program language was a set of synthetic computer instructions. Although these systems greatly simplified programming, the programmer was still plagued with the many details dictated by a computer.

Automatic coding languages of today are on the threshold of relieving the programmer of these details. The structure of these new languages is very much like English. By using a combination of English words and phrases to form sentences, the programmer now needs only to "describe" a procedure for the computer to follow. This procedure, together with a description of the data is then given to a special computer program for processing. The special program, commonly called a compiler, translates the English problem description and generates a program of computer instructions.

Such a compiler is provided for the GE 225. Its General Compiler evolved from two noteworthy language efforts - the Common Business Oriented Language (COBOL) and the Algorithmic Language (ALGOL). Both languages were developed by voluntary committees of computer manufacturers and users and reflect the recent trend toward "common" compiler languages.

The language first available with the General Compiler is based primarily on COBOL, since COBOL satisfies the needs of a broad spectrum of data processing applications. To accommodate the demands of more technical applications, Boolean expressions, floating point arithmetic, and the ability to express equations were incorporated into the format of COBOL. Therefore, one may say that the present version of the General Compiler can accept programs written in one, two, or in a combination of two languages.

Those programmers familiar with COBOL recognize that it is well suited for creating and reporting information contained in data files. In contrast, ALGOL provides an excellent means for expressing the mathematics and logic associated with scientific applications. Recent investigations by the Integrated Systems Project (ISP) of General Electric's Manufacturing Services uncovered an area of applications which require neither extensive data file processing nor profound mathematics, but rather an unwieldly number of sequential decisions.

To cope effectively with these decisions the ISP team devised a tabular language. The purpose of this language was to depict, by means of tables, the decisions encountered in the information flow of a business system. The new language was appropriately named TABSOL for Tabular Systems Oriented Language. Since its creation, TABSOL has been used by many departments of General Electric to analyze and solve problems in product engineering, manufacturing methods, cost accounting, and production control. The application of decision tables is continually growing. Recent studies show that they provide a concise method for supporting the logic of other data processing applications. For example, decision tables may be used to specify a transfer vector associated with the values of one or more fields, to control the printing of detail and summary lines of a report, or to interrogate the sort keys in a multi-file run. At the Computer Department we have found decision tables a valuable tool in designing and implementing the General Compiler.

Decision tables represent a third language for the General Compiler. They may be used by themselves or in conjunction with other features of the compiler language. The specifications outlined in this manual pertain mainly to the table entries and imply and require a knowledge of the General Compiler. Therefore, this manual should be used as a supplement to the GE 225 General Compiler Manual, CPB-123 (5.5M10-60).

II DECISION TABLE FORMAT

The format of a decision table is given in Fig. 1. In concept, a table is an array of blocks divided into four quadrants by a pair of double lines. The vertical double line separates the decisions or "conditions" on the left from the "actions" on the right. The horizontal double line isolates variables from associated operands which will appear in the blocks and rows below. A condition then is a relation between a variable appearing in a primary block and an operand appearing in a corresponding secondary block. For example, we may write AGE in primary block 1 and EQ 26 in secondary block 1. In doing this, we are stating a condition. Verbally, we are asking "if age equals 26". An action, on the other hand, is a statement of what is to be done. By writing AGE in a primary action block and 26 in its associated secondary block, we are stating that "the value 26 is to be assigned to age".

It is interesting to note, at this point, the English interpretation given to the vertical lines. The left-most line may be thought of as representing the word IF. Those lines to the left of the vertical double line may be taken to mean AND: the vertical double line itself the word THEN. Since actions are sequential entities, the lines separating them may be interpreted as semicolons and the right-most line, which actually terminates the actions, as a period. With this in mind, each secondary row becomes an English sentence. For example, each row now reads:

> "IF condition -1 is satisfied AND condition-2 is satisfied AND . . . AND condition-k is satisfied THEN perform action-1; action-2; . . . ; action m."

If any condition within a row is not satisfied, the next row is evaluated and so on until all the rows are depleted. When this happens the table is said to have "no solution". The table is considered "solved" when all the conditions of a row are satisfied and their associated actions performed.

Before considering the conventions used to formulate conditions and actions, an example may help develop insight into the nature of decision tables and the manner in which they may be used with the General Compiler. In the example of Fig. 2 we are searching a master employee file (recorded on magnetic tape) to determine the number of male employees who fall into the following job categories.

Job Level	Years Experience	Title
6	2	Programmer
7	3	Programmer or Analyst
8	More than 3	Analyst
9	More than 4	Analyst or Sr. Analyst
10	More than 4	Sr. Analyst

For each employee we find having these qualifications, we are to write his department number, name, title, level, and experience on the computer's typewriter. At the end of the run the total for each category is also typed on the typewriter.

The core of this problem is the decision that must be made on the information stored in the records of the master file. These decisions are conveniently expressed above in narrative form. With only minor alteration this form becomes the program statement of our problem. The table and sentences are punched into 80-column cards exactly as they appear in Fig. 2. When this is done they may be given directly to the compiler for processing.

As illustrated in our example, General Compiler sentences may be used to support the logic of the table. These sentences accomplish the following:

OPEN - Declares that the MASTER~FILE is input and validates its tape labels.

READ - Delivers the next record from the MASTER~FILE and tests for an end-of-file sentinel. When this sentinel is detected, sequential program execution is interrupted and control passes to the portion of the program labeled END~RUN.

IF - Eliminates those data records which contain information about female employees. The word FEMALE (also PROGRAMMER, ANALYST, and SR~ANALYST used in the table) represents a special kind of condition and will be explained later in the manual.

EXPERIENCE = Calculates the employees' total experience and assigns the value to the field named EXPERIENCE.

The word TABLE informs the compiler that it must process a decision table; EXAMPLE is a name or label which was given to the table. The size of the table is stated next by giving the number of conditions, actions, and rows contained in the table. This information is used only by the compiler and is not executed by the compiled program.

Table execution begins at row 1 (sequence number 40). Using our narrative definition of a table, row 1 is interpreted as follows:

> "IF the job LEVEL field equals (EQ) 6 AND the EXPERIENCE field equals (EQ) 2 years AND the employee's title is PROGRAMMER THEN assign the value 1 to the subscript I; GO TO the part of the program having the label TYPE~OUT."

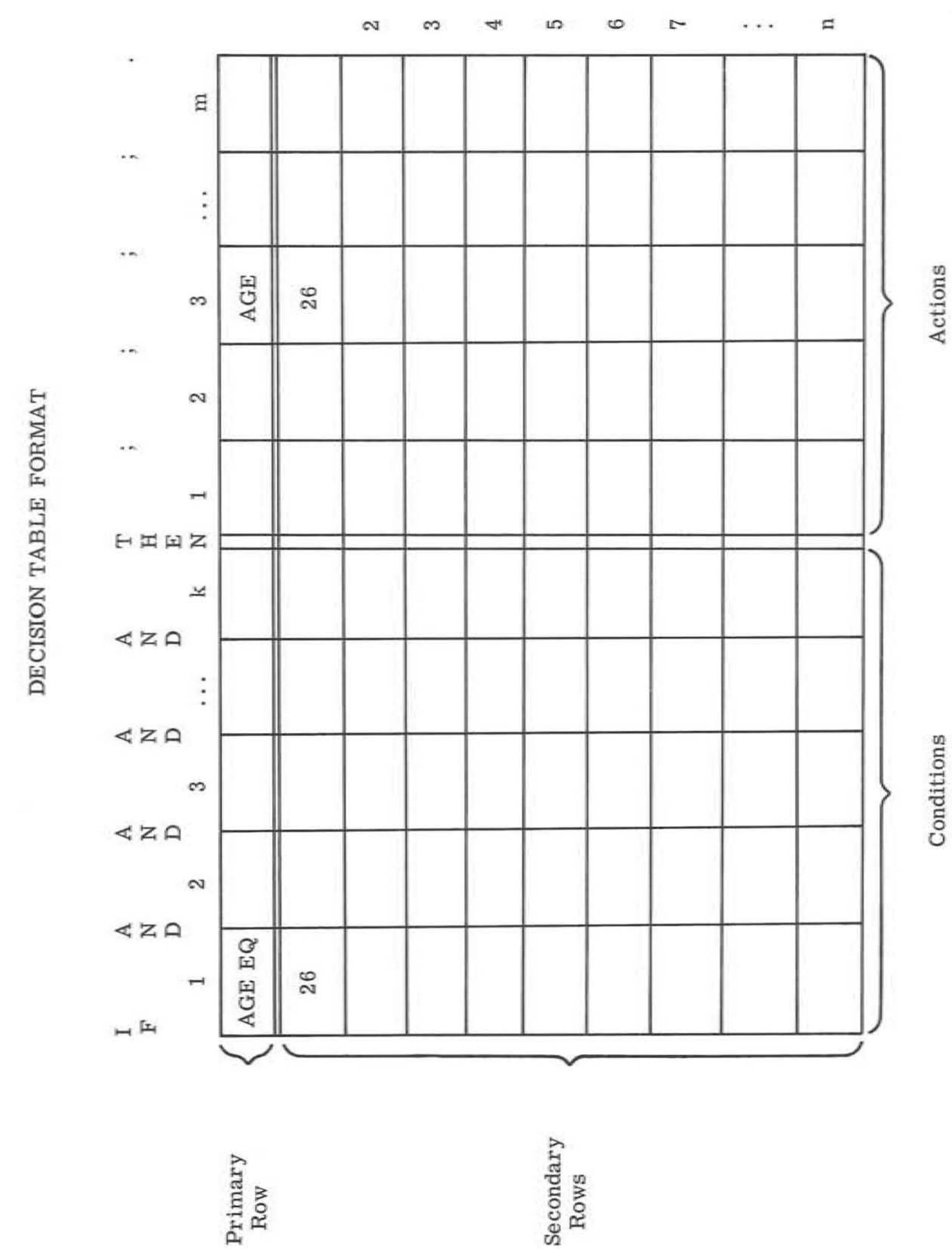


Figure 1

GENERAL ELECTRIC

GENERAL COMPILER
SENTENCE FORM

Computer Department, Phoenix, Arizona CA-13 (10-60)

PROGRAM Sample Decision Table

```
  5  PROCEDURE DIVISION
 10      OPEN INPUT MASTER~FILE
 15  GET~RECORD. READ MASTER~FILE RECORD IF END FILE GO TO END~RUN
 20      IF FEMALE GO TO GET~RECORD
 25      EXPERIENCE = 60 - YR~EMPLOYED + PREV~EXP.

 30  TABLE EXAMPLE. 3 CONDITIONS 2 ACTIONS 5 ROWS.

 35   LEVEL  EQ  EXPERIENCE                        I   GO TO
 40      6         EQ 2   PROGRAMMER               1   TYPEOUT
 45      7         EQ 3   PROGRAMMER OR ANALYST    2
 50      8         GR 3   ANALYST                  3
 55      9         GR 4   ANALYST OR SR~ANALYST    4
 60     10         GR 4   SR~ANALYST               5

 65      GO TO GET~RECORD.
 70  TYPE~OUT. WRITE DEPARTMENT NAME TITLE LEVEL EXPERIENCE ON TYPEWRITER
 75      TOTAL(I) = TOTAL(I) + 1.
 80      GO TO GET~RECORD.
 85  END~RUN. CLOSE MASTER~FILE
 90      WRITE TOTAL(1) TOTAL(2) TOTAL(3) TOTAL(4) TOTAL(5) ON TYPEWRITER.
 95      STOP "END RUN".
```

Figure 2

If one of these conditions cannot be satisfied, row 2 is evaluated starting again with the left-most condition. Sequential execution of the rows continues until either all conditions in a given row are satisfied or all rows are exhausted. When the latter situation occurs, the sentence immediately following the table is executed. Proceeding from here the sentences in our example accomplish the following:

GO - Interrupts sequential program execution and passes control to the part of the program labeled GET~RECORD.

WRITE - Writes the current contents of the DEPARTMENT, NAME, TITLE, LEVEL, and EXPERIENCE fields on the computer's typewriter.

CLOSE - Rewinds the MASTER~FILE and performs the file's closing conventions.

STOP - Terminates processing and writes the words END RUN on the typewriter.

By General Compiler standards this example represents relatively simple conditions and actions. In formulating these entries, the programmer may take full advantage of the compiler's capabilities. The remaining sections of this manual are devoted to defining the conventions and manner in which conditions and actions may be formed and entered in tables.

III BASIC CONCEPTS

Since decision tables are used in conjunction with the General Compiler language, we must first look at the foundations of this language before considering the counterparts that may appear in a table. The compiler's language, like most natural languages, is a body of words and a set of conventions for combining these words to express meanings. Its structure or "syntax" closely resembles that of English grammer, and its body of words may be appropriately termed a "vocabulary". The purpose of this section is to show how words are formed and how they may be used to express a desired meaning.

Characters

The basic units of our language are the characters used to form words and symbols. The character set includes the letters of the alphabet (A, B, C, ..., Z), the numerals (0, 1, 2, ..., 9), and the special characters shown in Fig. 3. Special characters are presented in more detail as they are encountered in the manual.

Words

The words of a typical General Compiler program fall into one of two categories: the vocabulary used by the compiler and the vocabulary used by the programmer. The programmer's vocabulary will consist mostly of arbitrary names given to his data. The compiler's vocabulary, on the other hand, is predetermined and explicitly defined in this manual. Since the compiler, by nature of its design, is a mistrusting mechanism, the programmer must define the words he uses too. This is done, not by writing a manual, but instead by merely filling out a data description form. Once these "data names" are defined, they may be filed either on 80-column punched cards or on magnetic tape and used over and over again. The data description file then is a "dictionary" since it contains the definitions of the words used by the programmer.

Our two categories of words may be illustrated by the following sentence taken from the program example given in Fig. 2.

GET~RECORD. <u>READ</u> MASTER~FILE
<u>RECORD</u> <u>IF</u> <u>END</u> <u>FILE</u> <u>GO</u> <u>TO</u> END~RUN.

Here, the words READ, RECORD, IF, END, FILE, GO, and TO belong to the vocabulary of the compiler; whereas, the words GET~RECORD, MASTER~FILE, and END~RUN belong to the programmer's vocabulary. The compiler will assume that MASTER~FILE is a data name due to its position in the sentence. It will then search the data description to verify this assumption and to determine the characteristics depicted by this word. Not finding a match in the data description results in an error message typed on the computer's typewriter. Due to their position in the program, the words GET~RECORD and END~RUN will be interpreted as sentence names. Once again, the compiler will attempt to verify its findings by checking each transfer to make certain that they lead to properly defined sentence names. The consequence of an undefined sentence name is likewise an error message on the computer's typewriter. The compatibility checks mentioned here are only two of many which the compiler performs to insure unquestionable results in the programs which it creates.

Formation of Names

As previously mentioned, data names are words representing data (files, records, fields, elements, constants, arrays of values, etc.) and are arbitrarily assigned by the programmer. They are formed from the following characters.

Letters	A, B, C, ..., Z
Numerals	0, 1, 2, ..., 9
Hyphen	~

The programmer should choose data names that

1. Do not exceed 12 characters,
2. Do contain at least one letter,
3. Do not begin or end with a hyphen.

All data names should be recorded and their characteristics described on the compiler's data description form. The programmer also should be careful not to use the compiler's vocabulary as data names.

In addition to data names, the programmer is free to name sentences, tables, and other "procedures" in his program. These names are formed like data names. Since procedure names are judged from their position in the program, they may be formed from the numerals 0 through 9 in addition to combinations of letters, numerals, and the hyphen.

Constants

The values associated with data names generally change during the actual running of a compiled program. It is for this reason that they are sometimes called "variables". A constant, as opposed to a variable, is a specific value and does not change within the scope of a program. Constants may be one of two kinds: a literal, or a named constant.

A literal is a value itself rather than a name given to a value. Literals may be numeric, alphabetic, or alphanumeric -- i.e., composed from the character set of the computer. All non-numeric literals should be enclosed in quotation marks (") to avoid having the compiler confuse them with data names. The conventions for forming literals are the following:

SPECIAL CHARACTERS

Character	Meaning	Card Code
Δ	Space or blank	Space
.	Period - Decimal point	12-3-8
,	Comma	0-3-8
"	Quotation Mark	3-8
~	Hyphen	5-8
(	Left Parenthesis	0-5-8
)	Right Parenthesis	0-6-8
+	Addition	12
-	Subtraction - Minus Sign	11
*	Multiplication	11-4-8
/	Division	0-1
=	Assignment	6-8
I	Vertical Table Line	12-4-8

Figure 3

1. Non-numeric literals are limited to 30 characters, excluding the quotation marks.

2. A numeric literal not enclosed in quotation marks is assumed to be a number. Numbers may contain not more than one decimal point and a minus sign. Unsigned numbers are considered positive. Excluding decimal points and minus signs, numbers must not exceed 11 decimal digits.

3. Numbers may be treated as floating point by writing them as a power of ten -- i.e., a number or decimal fraction followed by a power of ten exponent. For example, the number 230100 might be written as 2.301E5 which is equivalent to 2.301 multiplied by 10^5. The exponent part, indicated by the letter E, may contain a minus sign to show a negative exponent. The value of the exponent part is limited to ±75. Excluding the decimal point, the minus sign, and the letter E, the fractional part of a power of ten number must not exceed nine decimal digits. To distinguish data names from floating point numbers, data names should not be formed from only the numerals and the letter E.

4. An alphanumeric literal may not contain an embedded quotation mark since the enclosing quotation marks are used to delimit the size and content of the literal.

Subscripts

Subscripts provide a convenient method to reference individual values contained in a list or in an array of values. The variable, I, employed in the decision table of Fig. 2 is a subscript used just for this purpose. Since five totals are to be accumulated, one name was assigned to all five, namely, the data name TOTAL. Whenever reference was made to a particular total, the data name TOTAL was followed by the subscript I. This is illustrated in the expression

TOTAL (I) = TOTAL (I) + 1

and the sentence which prints all five totals on the typewriter. From this example, it follows that subscripts, like data, may be given names. In fact, the same rules that govern forming data names apply to naming subscripts.

Since subscripting is a positional notation, the range of any subscript is limited to the values 1, 2, 3, . . ., n (where n is the maximum number of values in a list). This does not mean that subscripts are limited only to integers. If a subscript is not defined as an integer by means of the data division, the compiler will automatically provide coding to truncate its value to an integer. Furthermore, subscripts are not restricted to a single variable name.

Arithmetic expressions may also be used as subscripts. For example:

RATE (P+1)

K ((X-3) * P **3)

A (J - 3 + Q * P)

are legitimate forms of subscripts.

Up until now, only one-dimensional subscripting was considered. Values in multi-dimensioned arrays may also be referenced by subscripts. For example, an array in which values are ordered

$A_{11}\ A_{12}\ A_{13}\ A_{14}\ A_{15}$

$A_{21}\ A_{22}\ A_{23}\ A_{24}\ A_{25}$

$A_{31}\ A_{32}\ A_{33}\ A_{34}\ A_{35}$

$A_{41}\ A_{42}\ A_{43}\ A_{44}\ A_{45}$

$A_{51}\ A_{52}\ A_{53}\ A_{54}\ A_{55}$

might be subscripted as A (J, K), where K is the columnar subscript and J the row. To refer to value A_{35}, J would have to equal 3 and K equal 5.

Preceding examples show that subscripts are enclosed in parenthesis and separated by commas. This notation permits the compiler to distinguish subscripts from other elements in the language.

Truth-Values

There is a class of variables which, through either usage or definition, may assume only the numerals 1 or 0. The value 1 is said to be their true state and the value 0 their false state. The words END FILE of the READ sentence in Fig. 2 is such a variable. When the OPEN sentence is executed, END FILE is set to its false state and remains so set until the end-file condition is encountered. At this time it is set to its true state.

Variables having truth values are termed "True-False" variables. END FILE is convenience provided by the compiler; the programmer may also formulate his own true-false variables by merely listing them under the heading TRUE-FALSE in the data division. They may be named according to the rules given for data names.

Arithmetic Expressions

Arithmetic expressions are rules for computing numerical values. They are formed from variables, numbers, functions, and symbols representing addition, subtraction, multiplication, division, and exponentiation. For example, in the expression

REG~HRS * 2.50 + OT~HRS * 3.75

REG~HRS and OT~HRS are variables; 2.50 and 3.75 numbers; and + and * symbols for addition and multiplication. If REG~HRS were 40 and OT~HRS were 4, the expression becomes 40 * 2.50 + 4 * 3.75 and after performing the arithmetic, reduces to the value 115.00. To save this value, a programmer might write

GROSS~PAY = REG~HRS * 2.50 + OT~HRS * 3.75

The presence of the = symbol tells the compiler to assign 115.00 to the variable GROSS~PAY. When expressions are written in this form, they are called "assignment statements".

The arithmetic permitted in an expression is stated by the following symbols:

Symbol	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

In addition to arithmetic, the following mathematical functions may be used:

Symbol	Function
SIN	Sine
COS	Cosine
ATAN	Arctangent
SQRT	Square Root
EXP	Exponential
LOG	Common Logarith
LN	Natural Logarithm
ABS	Absolute Value

Arithmetic expressions are evaluated from left to right according to the following priority:

1. Exponentiation and Functions
2. Multiplication and Division
3. Addition and Subtraction

Parentheses may be used to establish a precedence other than the one above. When they are used, the evaluation is performed from the innermost to the outermost pair but still from left to right within a given pair.

Relational Expressions

A relational expression is a statement of magnitude between two values. For example, FICA GR 144.00 is a comparison between the variable FICA and the numbers 144.00. The symbol GR stands for the relation "greater than". Other relations may be stipulated by

Symbol	Relation
EQ	Equal to
GR	Greater than
LS	Less than
NEQ	Not equal to
NGR	Not greater than
NLS	Not less than

To have meaning, relational expressions are stated as conditions. The expression FICA GR 144.00 tells us nothing. However, when it is written as

IF FICA GR 144.00, GO TO ADJUST~PAY.

we know immediately what is intended. By definition then, relational expressions are conditions and when evaluated always give a truth-value.

Relational expressions may be explicitly stated or implied. FICA GR 144.00 is an explicit statement of magnitude. In the program example of Fig. 2, implied relations were stated by the words FEMALE, PROGRAMMER, ANALYST, and SR~ANALYST. An implied expression is formed by giving a name to a value, a range of values, or to a series of values and ranges. Once the name and its values are defined in the data division, it may be used to mean its associated values. Implied relations are termed "condition-names" since a name is given to a condition, i.e., a value, of a variable. The variable from which the value is taken is called a "conditional variable". Therefore, writing PROGRAMMER (Fig. 2) in a decision table block is the same as writing an expression which will compare the TITLE field with the value associated with the title, programmer.

Logical Expressions

Logical expressions provide a convenient method for obtaining truth-values. They are formed by combining true-false variables and relational expressions with the logical operators AND, OR, and NOT. The expression (Fig. 2)

PROGRAMMER OR ANALYST

is a logical expression which is true when an employee's TITLE field indicates that he is either a programmer or an analyst.

If p and q are a combination of true-false variables, relational expressions, or logical expressions, their truth-value is obtained according to the following:

p	F	F	T	T
q	F	T	F	T
NOT p	T	T	F	F
p AND q	F	F	F	T
p OR q	F	T	T	T

Logical expressions are evaluated from left to right with the logical operator AND having precedence over the OR. Parentheses may be used for grouping or establishing a precedence of evaluation other than the one mentioned previously. When they are used, the evaluation proceeds from left to right from the innermost pair to the outermost pair.

IV TABLE ENTRIES

The previous section outlined the elements of the General Compiler language and briefly showed how they might be used. In the introduction, it was mentioned that these same elements may be employed within the blocks of decision tables. The purpose of this section is to show how this may be done.

Formation of Conditions

By definition, a condition is a relation between a primary block entry and some corresponding secondary block entry. A condition, like a relational expression, may be either true or false. From this definition, a condition may be either a relational expression, a logical expression, or a true-false variable since these are the only elements that yield a truth-value.

The formats noted below show how these expressions may be split between primary and secondary blocks to form conditions. In these examples, the word "operand" stands for either a variable (data name or subscripted data name), a constant (literal or named constant), or an arithmetic expression. The word "relation" signifies one of the relational operators - EQ, GR, LS, NEQ, NGR, or NLS. Since arithmetic expressions may be operands of relational expressions and relational expressions operands of logical expressions, it necessarily follows that arithmetic expressions may appear in logical expressions.

Format	Example
Operand-1 Relation / Operand-2	LEVEL EQ / 10
Operand-1 / Relation Operand-2	EXPERIENCE / GR 4
Operand-1 Relation / Operand-2 OR Operand-3	TOTAL (I) NLS / PT(1) OR PT(2) OR PT(3)
Operand-1 / Relation-1 Operand$_2$ OR Relation-2 Operand-3...	(X+Y) ** 3 / GR P+1 OR LS Q(I)
No Entry / Condition-name	/ PROGRAMMER
NOT / Condition-name	NOT / FEMALE
No Entry / True-False Variable	/ REQ 1
NOT / True-False Variable	NOT / END INVENTORY FILE
No Entry / Logical Expression	/ PROGRAMMER OR ANALYST
NOT / Logical Expression	NOT / X GR Y OR X LS (Z+1)

Formation of Actions

Actions are statements of the things to be done when all the conditions of a row are satisfied. The scope of an action may be one of three kinds: implied assignment, procedural, or input-output. The only action presented so far was assignment. The other two are extensions of General Compiler sentences and will be mentioned here only briefly. The compiler manual should be consulted for a more detailed presentation.

1. Value Assignment. Value assignment is an implied function between associated primary and secondary block entries. By placing a data name in a primary block and some number in a secondary block, for example, I and 1 of Fig. 2, the compiler automatically produces coding to assign the number to the data name. In the case of our example, 1 is assigned to the subscript I. Other examples of value assignment are given below. In these formats the word variable implies either a data name or a subscripted data name and the word constant either a literal or a named constant.

Format	Example
Variable / Constant	I / I

Format	Example
Constant	"COPPER"
Variable	MATERIAL
Variable	ALPHA (I, J, K)
Arithmetic Expression	SIN THETA + (X/P) **2
Arithmetic Expression	PI * R**2
Variable	AREA 1
True-False Variable	SWITCH 7
Truth-Value 1 or 0	1
Truth-Value 1 or 0	0
True-False Variable	BETA REQ

2. Procedural Actions: Procedural actions provide the means for interrupting the normal execution sequence of a table. Any of the following compiler verbs may be used for this purpose.

GO TO
PERFORM
STOP

The GO verb stipulates an unconditional transfer to a specified part of the table or program. Its destination may be a sentence name, table name, or the row number of a particular table. The format of the GO entry is as follows:

Format	Example
GO TO	GO TO
Sentence Name	TYPE~OUT
GO TO	GO TO
Table name	TABLE 23
GO TO	GO TO
Row of Table	ROW 7 TABLE BETA

The other form of a procedural control is the PERFORM verb. The PERFORM specifies a transfer to some destination, the execution of a table or a set of sentences at that destination, and a return to the action block following the PERFORM. The sentences or tables acted upon are by definition a "closed procedure" - i.e., they have a single entrance point and a defined exit point. Conventions for writing closed procedures are given in the next section. Legitimate forms of the PERFORM are

Format	Example
PERFORM	PERFORM
Sentence Name	GROSS~PAY
PERFORM	PERFORM
Table Name	ERROR TABLE

The STOP verb may also be used as an action. It may be placed in either a primary or secondary block. When it is used, no other action may appear with it in the same action column. The STOP terminates processing temporarily or permanently according to what action is taken at the computer's console.

3. Input-Output Actions: Input and output actions are compiler verbs that control the flow of data to and from the computer. They read, write, and validate tape labels of data files assigned to peripheral input-output devices. When data files are referred to from an action block, they must be defined according to the environment and data division specifications listed in the General Compiler manual. The formats of input-output actions are illustrated by the following:

Format	Example
READ	READ
File Name	MASTER~FILE
OPEN INPUT or OUTPUT	OPEN INPUT
File Name	MASTER~FILE
CLOSE	CLOSE
File Name	MASTER~FILE

Format	Example
File Name	MASTER~FILE
READ, CLOSE, or OPEN vrb	READ
WRITE	WRITE
Record Name	DETAIL~LINE
Record Name	TRANSACTION
WRITE	WRITE

The Skip and Repeat Operators

The skip operator makes it possible to show that a condition or action is not to take part in the evaluation of a row. This is done by placing a hyphen (~) in the concerned condition or action block. The compiler then will skip this block and proceed to the next.

The repeat operator is a shorthand method to indicate that a condition or action in the block above is repeated. This is shown by entering a ditto mark (") in the block below the one that is to be repeated. This notation was used with the GO TO action in the sample table of Fig. 2.

V THE TABLE AS A PROGRAM

Up until now, only components of tables were presented. It was learned in Section II that General Compiler sentences could be used to support the conditions and actions of tables, and the preceding section mentioned tables as closed procedures. This section relates these topics to tables and tables to compiler programs.

Block Conventions for Writing Expressions

1. Words, abbreviations, and symbols of the compiler's vocabulary should not be used as names. They may be combined with other characters to form names.

2. The words in an expression should be separated by at least one space. More than one space is permitted. The space separator is optional if the words are bound by

+ - * / ** ( ) " = , |

3. Subscripts should be enclosed in parentheses. They may be written adjacent to (without a space separator) or apart (with space separators) from their associated data names. Individual subscripts in a list of subscripts should be separated by commas.

4. When two arithmetic expressions appear side by side as in a series, they should be separated by commas.

5. All columns of a table should be bound by the vertical table line, | (12-4-8 punch).

6. The skip and repeat symbols, and ", should be the only entry, other than spaces in a block.

Conventions for Placing a Table in a Program

1. Tables are written on the General Compiler Sentence Form.

2. A table is preceded by the word TABLE. Naming tables is optional. When a table is given a name, the name may precede or follow the word TABLE. The word

TABLE,
name, TABLE, or
TABLE name

are followed by a period.

3. The table's size is given next and is placed on the same line as the table's name. The size may be written in one of two ways:

kkk CONDITIONS mmm ACTIONS nnn ROWS.

or

(kkk, mmm, nnn).

Both forms are terminated by a period. The order of writing the number of conditions, actions, and rows is optional in the first case since each can be identified. However, order is important in the second form since the compiler interprets the first number enclosed in parentheses as the number of conditions, the second as actions, and the third as rows. Conditions, actions, and rows are numbered sequentially beginning with 1. Row 1 is the first secondary row; the primary row is not counted in the row count.

4. The double vertical line that separates conditions from actions may be represented by one or two 12-4-5 punches.

5. The size of each block may vary from column to column and row to row.

6. The only limit on the size of a table is row width. Since the compiler prints a listing of compilation, the recommended row width is 120 characters including card sequence number. Maximum row width is 1200 characters. Since the table form is an image of an 80-column punched card, a hyphen (~) is placed in column 7 of the form to show that a row extends to more than one card. In this case, no table column may be split across cards. Each card is to contain a sequence number to insure proper card order. When rows exceed one card, the sequence number of the first card only is printed on the listing. Sequence numbers of succeeding cards are stripped out. The row is then printed as a multiple of 120 characters with an integral number of table columns per 120 characters.

7. Expressions too long or complex to be written in blocks may be written after the table's name and size and be executed from the table by means of the PERFORM verb. In addition to expressions, any General Compiler sentence may be used and executed in this manner. To indicate the start of the table the word BEGIN is to follow the list of expressions and sentences. When used, General Compiler sentences may not appear between BEGIN and the primary row of the table. This format may be illustrated by the following:

TABLE name. kkk CONDITIONS nnn ACTIONS nnn ROWS.

. . . General Compiler Sentences and Expressions - May be executed only from the confines of the table.

BEGIN

DECISION	TABLE

Closed Procedures

Fig. 4 outlines the format of a closed procedure. By definition a closed procedure may be acted on only by the PERFORM verb. It contains one entrance

point and one exit point. In Fig. 4 these are indicated by the words BEGIN and END TABLE name. BEGIN and END also act as sentence names and may be referred to from within the procedure body.

Expressions too long to be placed in the blocks of a table may be written in the procedure head and executed from the procedure body by means of the PERFORM verb. As such, they must be given names. In addition to expressions any General Compiler sentence may be written in the head and executed accordingly.

The procedure body contains the table. As shown in Fig. 4 compiler sentences may precede and follow the table. Execution is sequential starting with the sentence or table after the word BEGIN and proceeds until the exit END TABLE is reached. It is at this point that control is reverted to the PERFORM verb which originally referenced the procedure. Any unconditional transfer from within the procedure to the outside is undefined. However, PERFORM verbs in the body may reference other closed procedures.

Closed procedures are written apart from the main program.

DECISION TABLE AS A CLOSED PROCEDURE

procedure head:

TABLE name. kkk CONDITIONS mmm ACTIONS nnn ROWS.

... General Compiler Sentences and Expressions - May be executed from the confines of the decision table.

BEGIN. (Start of execution - entrance to procedure body)

procedure body:

... General Compiler Sentences and Expressions

Decision	Table

... General Compiler Sentences and Expressions

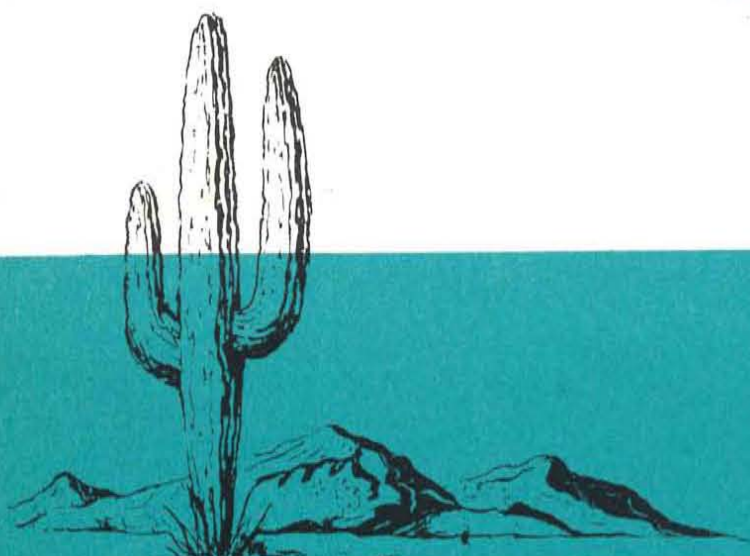
END TABLE name. (Exit of procedure body)

Figure 4

INFORMATION PROCESSING
AUTOMATED BY GENERAL ELECTRIC

Progress Is Our Most Important Product

GENERAL ELECTRIC



COMPUTER DEPARTMENT, PHOENIX, ARIZONA

CPB-147 (5M 3-61)