IBM

SYSTEM/360 COBOL

Writing Programs in COBOL

Reference Handbook

Programmed Instruction Course

IBM

SYSTEM/360 COBOL

Writing Programs in COBOL

Reference Handbook

Programmed Instruction Course

Minor Revision (July 1967)

This publication, R29-0211-2, is a reprint of Form R29-0211-1 incorporating minor editorial changes made on pages dated (7/67). The original publication is not obsoleted.

Copies of this publication can be obtained through IBM Branch Offices.
Address comments concerning the contents of this publication to:
IBM DPD Education Development, Education Center, Endicott, New York

PREFACE

This reference handbook provides information that will assist people in composing original COBOL programs. It is designed to be studied in conjunction with the Writing Programs in COBOL programmed instruction textbook (Form R29-0210).

The reader of these publications is expected to have read the previous course in this series, COBOL Program Fundamentals. The publications for that course are a programmed instruction textbook (Form R29-0205) and a reference handbook (Form R29-0206). Duplication of topics in the reference handbooks has been held to a minimum, with this handbook taking up where the previous one left off; therefore, the reader should also have a copy of the other handbook.

This publication is not intended to serve as a comprehensive reference manual for System/360 COBOL. Rather, it contains a selection of the most commonly used entry formats. In most cases, the formats have been abbreviated to reduce the number of details with which the programmer must concern himself. Particularly, entries and portions of entries that are used for processing non-sequential (random) files have been omitted. Thus, this book presents a subset of System/360 COBOL which is adequate for processing sequential files. Complete specifications for System/360 COBOL may be found in the reference manual, IBM Operating System/360 COBOL Language (Form C28-6516), or IBM System/360 Disk and Tape Operating Systems COBOL Language Specifications (Form C24-3433). The appropriate manual should be selected based upon the operating system that you plan to use.

ACKNOWLEDGEMENT

The following information is reprinted from COBOL Edition 1965, published by the Conference on Data Systems Languages (CODASYL), and printed by the U. S. Government Printing Office.

"Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas taken from this report as the basis for an instruction manual or for any other purpose is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention "COBOL" in acknowledgement of the source, but need not quote this entire section.

"COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

"No warranty, expressed or implied, is made by any contributor or by the COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

"Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

"The authors and copyright holders of the copyrighted material used herein

FLOW-MATIC (Trademark of Sperry Rand Corporation), Programming for the Univac (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals of similar publications".

TABLE OF CONTENTS

Program Sheet Format and Rules

Introduction to Entry Formats

Identification Division Entry Formats

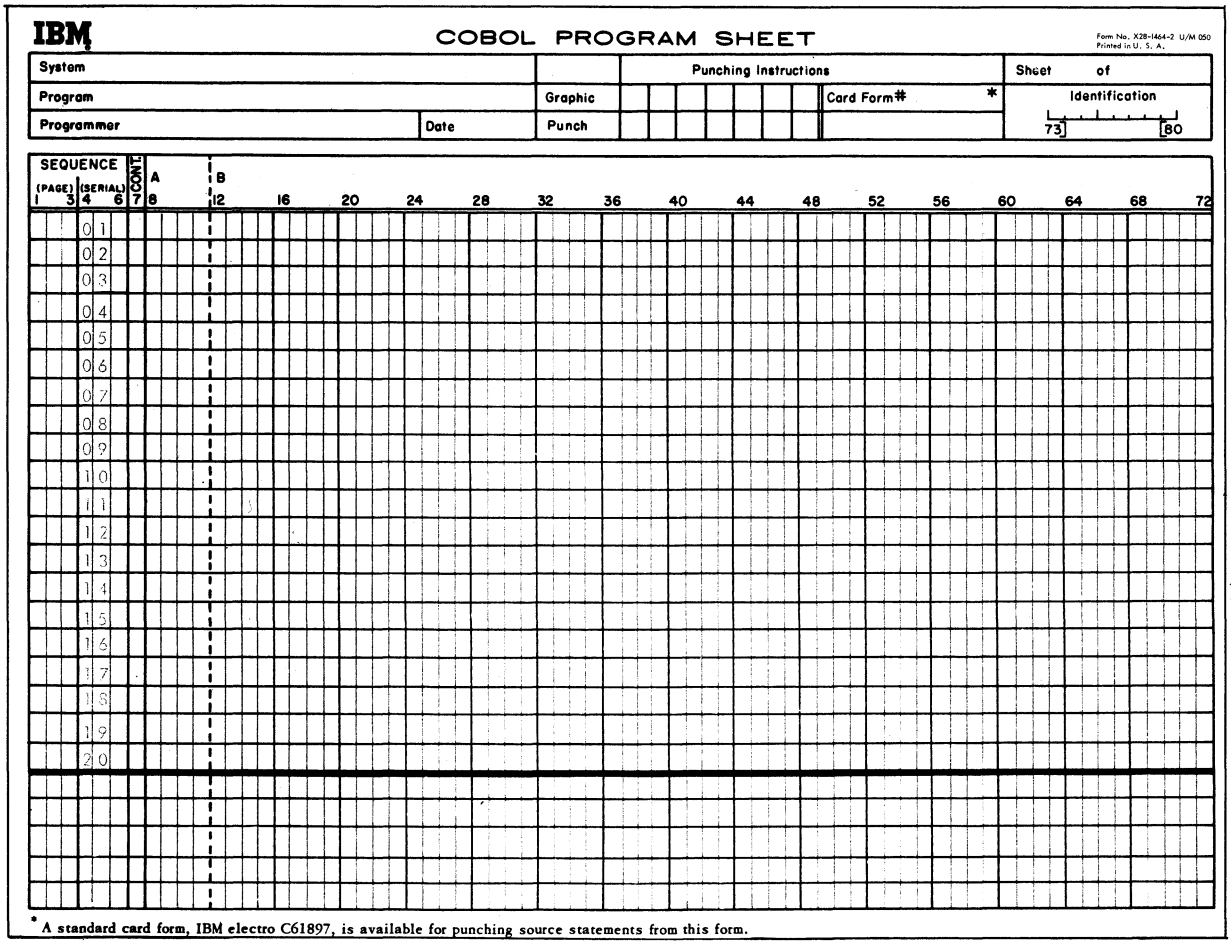
Environment Division Entry Formats

Data Division Entry Formats

Procedure Division Entry Formats

Program Sheet Format and Rules

PROGRAM SHEET FORMAT. The program sheet is designed to be the source document for keypunching COBOL program cards. A separate card is punched for each line that is used on the sheet. The sheet provides for 80 columns of information, corresponding to the 80 columns of a card. Information entered in the unnumbered boxes (labeled "System", "Program", etc.) at the top of the sheet is not punched.



* A standard card form, IBM electro C61897, is available for punching source statements from this form.

Columns 1-6 are used to number the lines of a program. Each succeeding line is given a higher number. Optionally, these columns may be left blank. Columns 4-6 are prenumbered for your convenience.

Column 7 is used for a hyphen to signify continuation of non-numeric literals.

Columns 8-72 are used for program entries. These columns are grouped into two "margins" -- margin A (8-11) and margin B (12-72).

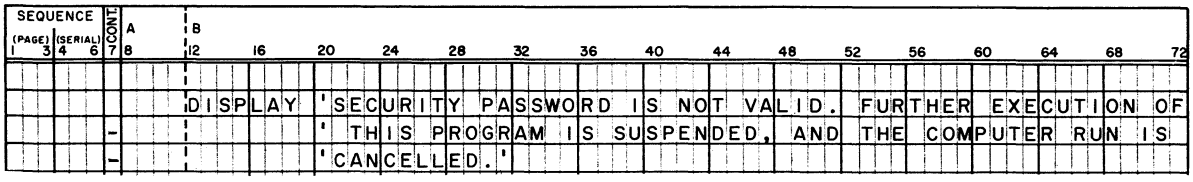
Columns 73-80 are used for the name of the program. Optionally, these columns may be left blank.

HOW ELEMENTS ARE WRITTEN IN ENTRIES.

Elements written on program sheets. Each element in an entry must be written in full on one line.

Exception: A non-numeric literal may be written on two or more lines if necessary.

- Characters of the literal are written all the way to column 72, since any blanks left at the end of a continued line are considered part of the literal.
- On every continuation line, a hyphen is written in column 7, and an extra quotation mark is written in margin B, followed immediately by more characters of the literal.



Spacing between elements. Each element must be separated from the next element by at least one space. (Note: If an element ends in column 72, it is treated as if it were followed by a space.)

Exception: Certain symbols must be written directly next to other elements, with no spaces left between.

The following symbols must not be followed by a space:

- Left parenthesis
- Beginning quotation mark (However, a space may be the first character of the literal enclosed by the quotation marks.)

The following symbols must not be preceded by a space:

- Period used to end an entry
- Comma or semicolon used to punctuate an entry
- Right parenthesis
- Ending quotation mark (However, a space may be the last character of the literal enclosed by the quotation marks.)

RULES FOR PROGRAM ENTRIES.

Entries that begin in Margin A. The following entries are required to begin in margin A (columns 8-11); customarily they begin in column 8:

- Division headers (must always be written on a line by themselves)
- Section headers (generally must be written on a line by themselves)
- Paragraph headers (need not be written on a line by themselves)
- File descriptions (found in the Data division -- start with two-letter reserved words called "level indicators", such as FD; the level indicator must be written in margin A, but the rest of a file description entry must be written in margin B)
- Two special headers in the Procedure division -- DECLARATIVES and END DECLARATIVES (must be written on a line by themselves)

Level numbers found in item description entries (as distinct from the level indicators mentioned above) may be written in margin A if desired. They are not required to be written there, however, and are usually indented into margin B.

No other entries are permitted in margin A.

Entries that begin in margin B. All other entries are required to begin in margin B (anywhere in columns 12 through 72). In practice, these entries usually either begin in column 12, or are indented to a column whose number is a multiple of 4 -- 16, 20, 24 ... Every fourth column of the program sheet is conveniently marked by a heavier line and numbered at the top.

New line required. Any entry that is required to begin in margin A must be written on a new line. Also, if an entry is required to appear on a line by itself, the next entry must begin on a new line.

Spacing between entries. Except when the next entry in a program is required to begin on a new line, it may follow on the same line as the previous entry, but must be separated from it by at least one space.

However, it is not necessary to fill out each line with entries, and a frequent practice is to begin each entry on a new line. A short entry may take up only part of a line, and the rest of the line may be left blank. An entry that would fit on one line may, if desired, be spread over two or more lines. It is also permissible, and often desirable, to leave entire lines blank.

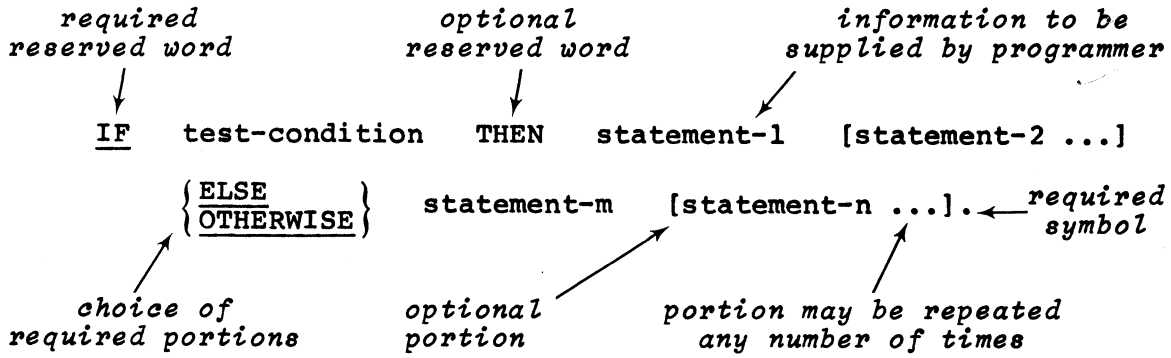
Continuation of entries. If an entry is too long to fit on one line, it is simply continued on the next line or lines. The continuation of an entry is always written in margin B, regardless of whether the entry began in margin B or in margin A. A hyphen is not written in column 7 to indicate continuation of an entry.

Introduction to Entry Formats

SYSTEM OF NOTATION USED TO DESCRIBE ENTRY FORMATS.

- Reserved words are printed entirely in capital letters.
- Reserved words that are required in the format are underlined. These words may be omitted only if the portion of the format containing them is itself optional.
- Optional reserved words are not underlined. These words are used only for the sake of readability, and may be included or omitted.
- Required symbols, such as periods and equal signs, are printed in the format, but not underlined. Optional symbols, such as commas and semicolons, are not printed in the format.
- Information to be supplied by the programmer is represented by words printed entirely in small letters. Unless it is in an optional portion of the format, this information is required (even though the words are not underlined).
- Optional portions of the format are enclosed in brackets []. Such portions may be used or omitted, as required for the program.
- A choice of optional portions is indicated by stacking them within brackets. This indicates that one -- or none -- may be used.
- A choice of required portions of the format is indicated by stacking them within braces { }. This indicates that one, and only one, must be used.
- Portions of the format that may be repeated any number of times are followed by three dots ... . When the dots follow a word, they apply only to that word. When the dots follow brackets or braces, they apply to all of the enclosed portion of the format.

Sample entry format.



Identification Division Entry Formats

Function To specify the name of the program; and optionally, to provide other information about the program.

Format

```
IDENTIFICATION DIVISION .
PROGRAM-ID .
    'program-name' .

[AUTHOR .
    entry ...]

[INSTALLATION .
    entry ...]

[DATE-WRITTEN .
    entry ...]

[DATE-COMPILED .
    entry ...]

[SECURITY .
    entry ...]

[REMARKS .
    entry ...]
```

Example

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
    'PAYROLL'.
REMARKS.
    PREPARES PAYCHECKS FOR ALL SALARIED
    EMPLOYEES. COMPUTES TAXES AND OTHER
    DEDUCTIONS. UPDATES ACCOUNTS TO SHOW
    CHANGES IN SALARY, EXEMPTIONS, ETC.
```

Notes Program-name must consist of a letter followed by no more than seven letters and/or digits. No special characters are allowed. The name must be enclosed in quotation marks.

Entries in the optional paragraphs may be made up of any words, numbers, or symbols, terminated by a period.

Environment Division Entry Formats

Function To specify the computers used to compile and execute the object program. Also, to assign each file to an input-output device, and to define special input-output conditions and techniques.

Format

```
[ ENVIRONMENT DIVISION .
  CONFIGURATION SECTION .
 [SOURCE-COMPUTER .
     IBM-360 [model-number].]
 [OBJECT-COMPUTER .
     IBM-360 [model-number].] ]

[ INPUT-OUTPUT SECTION .
  FILE-CONTROL .
     SELECT-entry ...

 [I-O-CONTROL .
    [APPLY-entry ...] ]
```

Example

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
    IBM-360 E30.
OBJECT-COMPUTER
    IBM-360 H40.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT SALES, ASSIGN 'SYS001' UTILITY.
    SELECT SUMMARY-OF-SALES,
        ASSIGN 'SYS002' UNIT-RECORD 1403.
I-O-CONTROL.
    APPLY END-OF-SHEET TO FORM-OVERFLOW
        ON SUMMARY-OF-SALES.
```

Notes Model-number consists of a letter to designate main core storage capacity, followed by the System/360 model number.

The Input-Output section must be included if there are any input or output files; hence, it is required in most programs. The I-O-Control paragraph may be omitted if no special techniques or conditions need to be defined in the program. The formats of the SELECT-entry and APPLY-entry are explained on the following pages.

Besides the APPLY-entry, the I-O-Control paragraph can contain entries that are not discussed in this book.

Function To assign a file to an input-output device.

Format

```
SELECT  file-name  ASSIGN TO  'external-file-name'

{UTILITY       }
{DIRECT-ACCESS } [device-number [UNIT ] ].
{UNIT-RECORD   }               [UNITS]
```

Example

```
SELECT INVENTORY-FILE,
    ASSIGN TO 'SYS005',
    DIRECT-ACCESS 2311 UNIT.
```

Notes This entry is written in the File-Control paragraph. A separate SELECT entry is required for each file.

The format shown is only suitable for sequential files. There is a different format of the SELECT entry, with several additional clauses, which is used for non-sequential files.

File-name is the programmer-supplied name by which the file will be referred to in the Data and Procedure divisions.

External-file-name is the name by which the file will be referred to on a job control card at the time that the object program is executed. It must consist of a letter followed by no more than seven letters and/or digits. No special characters are allowed. The name must be enclosed in quotation marks. If you are using DOS/TOS then it must be of the format 'SYSnnn' where nnn is a 3-digit number between 000 and 244.

UTILITY is specified for machines that can read and write data sequentially -- magnetic tape, disk, drum, and data cell devices. Allowable device-numbers for this class are 2400, 2301, 2302, 2311, 2321, and 2314.

DIRECT-ACCESS is specified for machines that can read and write data randomly -- disk, drum, and data cell devices. (Note that these machines also fall into the UTILITY class.) Allowable device-numbers are 2301, 2302, 2311, 2321, and 2314.

UNIT-RECORD is specified for printers and card read/punches. Allowable device-numbers are 1403, 1404, 1443, 1442R, 1442P, 2520R, 2520P, 2540R, and 2540P. (R indicates Reader; P indicates Punch.)

Function To define a condition name which will be used in an overflow test in the Procedure division. An overflow test determines whether the forms in a printer are at the end of a page. An APPLY entry is written only for an output file which is to be printed, and which overflows from sheet to sheet.

Format APPLY overflow-name TO FORM-OVERFLOW ON file-name .

Example

```
    APPLY NO-MORE-ROOM TO FORM-OVERFLOW
        ON MONTHLY-ACTIVITY-LISTING.
```

Notes This entry is written in the I-O-Control paragraph. It has other formats which serve different functions, such as specifying special input-output techniques for files; the other formats are not discussed in this book.

Overflow-name is the programmer-supplied name for the form-overflow condition. This entry is all that is required to define the meaning of this name; the name is not defined further in the Data division.

File-name is the programmer-supplied name of the file that is to be printed. This name must also appear in a SELECT entry in the File-Control paragraph; in that entry, the file must be assigned to a printer. In this way, the overflow-name is associated with the sensing of a 12-punch in the carriage control tape of a particular printer.

If the program involves more than one output file, a separate APPLY entry is written for each printed file for which an overflow test is needed.

Data Division Entry Formats

Function

To describe data files and the records in the files, as well as items in working-storage. The structure of each record is usually shown, with items described in the order in which they appear in the record, and with a breakdown of smaller items within larger items.

Format

```
DATA DIVISION .

[FILE SECTION .
 file-description
 record-description ...

[file-description
 record-description ...]    ...]

[WORKING-STORAGE SECTION .
[independent-item-description ...]
[record-description ...] ]
```

Example

```
DATA DIVISION.
FILE SECTION.
FD  TRANSACTION; BLOCK CONTAINS 25
    RECORDS; LABEL RECORDS ARE STANDARD;
    DATA RECORD IS TRANSACTION-RECORD.
01  TRANSACTION-RECORD.
    02  ACCOUNT-NUMBER, PICTURE X(10).
    02  TRANSACTION-CODE, PICTURE X.
    02  AMOUNT, PICTURE 9(6)V99.
WORKING-STORAGE SECTION.
77  PREVIOUS-NUMBER, PICTURE X(10).
01  MESSAGE.
    02  ACCOUNT, PICTURE X(10).
    02  FILLER, PICTURE XX, VALUE SPACE.
    02  TOTAL, PICTURE $ZZZ,ZZZ.99.
    02  FILLER, PICTURE XX, VALUE SPACE.
    02  COMMENTS, PICTURE X(30).
```

(Continued on next page)

Notes

Only the two most frequently used sections, the File section and the Working-Storage section, are shown in this format. There are other, less common, sections, which are not discussed in this book.

The File section is required in any program that processes input or output files; hence, it is required in most programs. It must contain a file description entry for every file named in a SELECT entry in the Environment division. Each file description entry must be followed by at least one record description; there must be a record description for each type of record in the file.

The Working-Storage section contains descriptions of constants and work areas. It may be omitted when constants and work areas are not used in a program.

Items in working storage that are not subdivided into smaller items are defined as independent items (level-77 items). The descriptions of all independent items must precede any record descriptions in the Working-Storage section.

Items in working storage that are subdivided into smaller items are defined as records. Record descriptions in the Working-Storage section have the same make-up as record descriptions in the File section. (Note, however, that records in the Working-Storage section are always group items, whereas records in the File section can be elementary items.)

Record descriptions are made up of item description entries.

The formats of file description and item description entries are given on the following pages.

FILE DESCRIPTION

Function To specify such characteristics of a file as its recording mode, the number of characters or records in a block, the number of characters in a record, whether there are label records, and what the names of its data records are.

Format

```
FD   file-name

[RECORDING MODE IS  mode]

[BLOCK CONTAINS  integer-1  {CHARACTERS}]
                            {RECORDS   }

[RECORD CONTAINS  [integer-2 TO]  integer-3  CHARACTERS]

{LABEL RECORD IS  }   {STANDARD}
{LABEL RECORDS ARE}   {OMITTED }
                      {data-name}

{DATA RECORD IS  }
{DATA RECORDS ARE}   record-name ...  .
```

Example

```
FD  OPINION-SURVEY,
    RECORDING MODE IS V,
    BLOCK CONTAINS 5 RECORDS,
    RECORD CONTAINS 120 TO 200
        CHARACTERS,
    LABEL RECORDS ARE STANDARD,
    DATA RECORD IS RESPONSE.
```

Notes There must be a file description entry for each input or output file. The above format is used for nearly all kinds of files; however, slightly different formats are used to describe files for the Report Writer and Sort features of COBOL. The other formats are not discussed in this book.

File-name is the programmer-supplied name of the file, and must be the same as a file-name specified in a SELECT entry in the Environment division.

(Continued on next page)

Notes

Mode must be either V, F, or U; however, the entire RECORDING MODE clause can be omitted when the mode is V. If each record is preceded by a control field that specifies the length of the record, then the mode is V. If all records in the file are the same length, and there are no record-length control fields, then the mode is F. If the records have variable lengths, and there are no record-length control fields, then the mode is U.

All integers called for in the format must be unsigned whole numbers.

Integer-1 specifies the number of characters in the longest block, or the number of the longest records that would form the longest block. The word CHARACTERS may be omitted when the number of characters is specified. The BLOCK CONTAINS clause is completely omitted when there is just one record in each block.

Integer-2 specifies the length of the shortest record, and integer-3 specifies the length of the longest record in the file. The RECORD CONTAINS clause may be omitted, since the compiler determines record lengths from the record descriptions.

The LABEL RECORD clause is required. STANDARD is specified for files with standard labels. OMITTED is specified for files with no labels or with non-standard labels. A data-name is specified for files that have user labels in addition to standard labels. Data-name is the programmer-supplied name of a storage area in which the user labels will be processed.

The DATA RECORD clause is also required. Record-name is the programmer-supplied name of a record in the file. The names of all records in the file must be written in this clause. Each record-name must also appear in a level-01 entry (in a record description) following the FD entry for its file.

RECORD STRUCTURE. Records are usually subdivided into smaller items. Some or all of those items may be further subdivided into still smaller items.

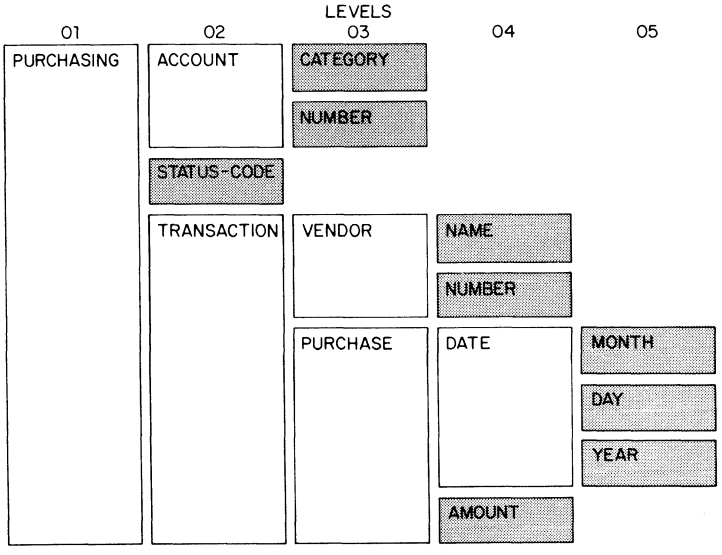
- When an item is subdivided, its parts fall into the next level of items within the record structure. Suppose, for instance, that DATE is subdivided into MONTH, DAY, and YEAR. In that case, DATE is at one level in the structure of the record, while MONTH, DAY, and YEAR are all at the next level.

- Each level is given a number, always beginning with 01 for the most inclusive item -- the record itself. Succeeding levels are given larger numbers, usually 02, 03, 04, etc.

- In subdividing a record, the numbers of the levels need not be consecutive, and numbers as large as 49 may be used. This gives the programmer some flexibility in assigning numbers; for example, he may choose to number the levels 01, 03, 05..., or 01, 05, 10, 15...

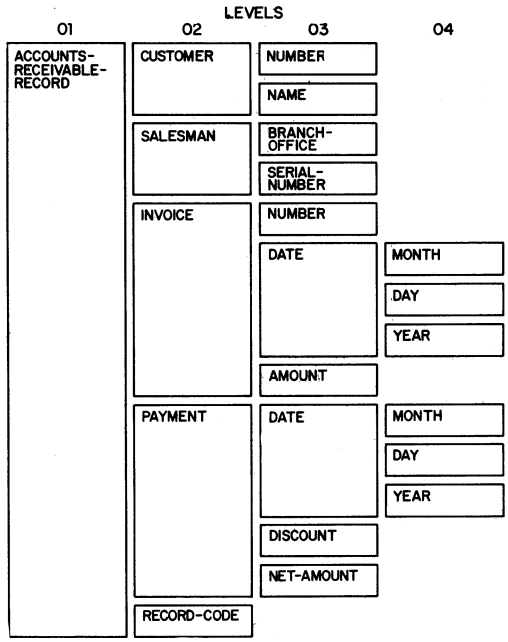
- It is not necessary for all items to be subdivided at every level; some may be subdivided, and others not. An item that is further subdivided is called a group item. An item that is not further subdivided is called an elementary item.

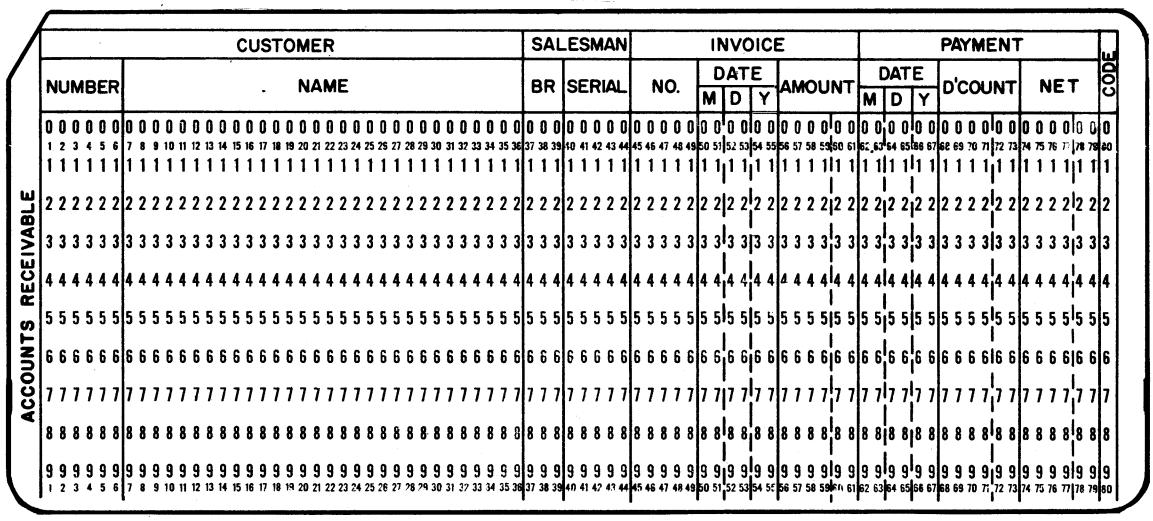
An illustration of levels of data items. In this schematic drawing, each box represents an item. The box for level 01 represents the area occupied by the entire record. At succeeding levels, the same area is subdivided into smaller items. In the case of one item (STATUS-CODE), subdivision stops at level 02; but subdivision of other items is carried as far as level 05. The boxes that represent elementary items are shaded in this drawing.



AN EXAMPLE OF A RECORD DESCRIPTION. In addition to a COBOL record description, a schematic drawing of the structure of the record, and the format of the physical record are shown. In this case, the record is shown on a card form, but it might just as well have been on any other recording medium.

```
01  ACCOUNTS-RECEIVABLE-RECORD.
    02  CUSTOMER.
        03  NUMBER                 PICTURE X(6).
        03  NAME                   PICTURE X(30).
    02  SALESMAN.
        03  BRANCH-OFFICE          PICTURE 999.
        03  SERIAL-NUMBER          PICTURE 9(5).
    02  INVOICE.
        03  NUMBER                 PICTURE 9(5).
        03  DATE.
            04  MONTH              PICTURE 99.
            04  DAY                PICTURE 99.
            04  YEAR               PICTURE 99.
        03  AMOUNT                 PICTURE 9(4)V99.
    02  PAYMENT.
        03  DATE.
            04  MONTH              PICTURE 99.
            04  DAY                PICTURE 99.
            04  YEAR               PICTURE 99.
        03  DISCOUNT               PICTURE 9(4)V99.
        03  NET-AMOUNT             PICTURE 9(4)V99.
    02  RECORD-CODE                PICTURE X.
```





DATA NAMES. It must be possible to identify every data item uniquely, in order to refer to each item individually in procedural statements. For this reason, items are given names; however, items can be referred to uniquely even when they do not have unique names.

Two ways of naming data items.

- A data item can have a unique name, that is, a name spelled differently from any other name in the program. In this case, the data name alone is enough to identify a specific data item.

- Two or more items can have the same name, that is, their names can be spelled exactly alike. In this case, each name must be "qualified" when it is used in the Procedure division, so there will be no doubt as to which data item is being referred to. (Qualification is explained below.)

Qualification of names. The name of a data item is "qualified" in the Procedure division by giving the names of one or more other data items that the item is part of. The names of the other items are called "qualifiers".

- Qualifiers are written after the name they qualify, with each qualifier preceded by the word OF or IN. (The illustration below shows excerpts from both the Data and Procedure divisions.)

```
    02  ORDER.
        03  AMOUNT  PICTURE 9(4)V99.
        03  DATE    PICTURE 9(6).

    MOVE CURRENT-DATE TO DATE OF ORDER.
```

- Whenever two or more items in a program have the same name, their names must be qualified, and each name must have at least one qualifier that is different from any possible qualifier of the others. The "highest" qualifier must be a unique name.

- In the File section, the highest possible qualifier is the name of a file; thus, it is permissible for two records to have the same name, but the names of files must be unique. In the Working-Storage section, a record name is the highest possible qualifier, so all record names in that section must be unique. Names of independent items must be unique, since they cannot be qualified.

Function To specify the level number and name of an item, and to describe its picture, value, and usage.

Format

```
level-number  {data-name}
              {FILLER  }

[PICTURE IS  {alpha-form  }]
             {an-form     }
             {numeric-form}
             {report-form }
             {fp-form     }

[VALUE IS  literal]

[USAGE IS  {DISPLAY             }  ].
           {COMPUTATIONAL       }
           {COMPUTATIONAL-1     }
           {COMPUTATIONAL-2     }
           {COMPUTATIONAL-3     }
```

Example

```
77  DEPARTMENT-TOTAL, PICTURE S9(5)V99,
    VALUE ZERO, COMPUTATIONAL-3.
```

Notes

This format is an abbreviation of the complete item description entry format, and shows only the three most most commonly used clauses: PICTURE, VALUE, and USAGE.

Level-number must be 01 if the item is a record; between 02 and 49, inclusive, if the item is part of a record; 77 if it is an independent item.

Picture is required in every description of an elementary item. It is forbidden in descriptions of group items.

A VALUE clause is permitted only in descriptions of elementary items, and only in the Working-Storage section. However, it is not allowed in descriptions of report items or external floating-point items. Literal must be a numeric literal if the item is numeric, and a non-numeric literal if the item is alphabetic or alphanumeric. The figurative constant ZERO may be used in place of either a numeric or a non-numeric literal, and SPACE may be used in place of a non-numeric literal.

A USAGE clause is allowed in both group and elementary item descriptions. This clause may be omitted if an item's usage is display, or if usage is specified for any group item that this item is part of (when this item has the same usage as the group item).

CONDITION-NAME

Function To assign a name to a particular value of an item.

Format 88 condition-name VALUE IS literal .

Example

```
        88 PARCEL-POST, VALUE 12.
```

Notes Condition-name is a programmer-supplied name that will be used as a test-condition (usually in an IF sentence) in the Procedure division.

The condition name is the name of a value of an item, not the name of an item; an item description entry is required to define the item itself. Level-88 entries must follow immediately after the item description entry for the item with which they are associated.

A condition name can only be associated with an elementary item.

Literal must be a numeric literal if the item has a numeric picture, and a non-numeric literal if the item has an alphanumeric or alphabetic picture. The figurative constant ZERO may be used in place of either a numeric or a non-numeric literal, and SPACE may be used in place of a non-numeric literal.

Procedure Division Entry Formats

Function

To specify the actions -- such as input-output, data movement, and arithmetic -- that are required to process the data. Also, to control the sequence in which those actions are carried out.

Format

```
PROCEDURE DIVISION .

procedure-name .
   sentence ...

[procedure-name .
   sentence ...] ...
```

Example

```
PROCEDURE DIVISION.
START-RUN.
    OPEN INPUT PRODUCTS, OUTPUT SAMPLES.
PROCESS-RECORDS.
    READ PRODUCTS; AT END,
        CLOSE PRODUCTS, SAMPLES;
        STOP RUN.
    IF PRODUCT-NUMBER > SAMPLE-BASE,
        MOVE PRODUCT-RECORD TO SAMPLE,
        WRITE SAMPLE.
    GO TO PROCESS-RECORDS.
```

Notes

The Procedure division contains one or more paragraphs. Paragraphs may also be grouped into sections.

Procedure-name is the programmer-supplied name of a paragraph or section.

A sentence is a procedural entry that consists of one or more statements. Each statement specifies an action to be taken, and begins with a verb, such as READ, MOVE, COMPUTE, or PERFORM.

The formats of the most commonly used procedural statements are given on the following pages.

Function — To obtain as many as 80 characters of data from the input device which has been specified as the "system logical input device". This input device might be a disk file, tape drive, card reader, etc.; however, the data must originally be punched into a card, and entered into the computer along with the job control cards at the time that the object program is executed -- hence the 80-character limit. This statement is used to get low-volume input, such as the current date, an initial serial number, or control totals; it is not used to read files of data.

Format — ACCEPT data-name

Example

```
    ACCEPT INITIALIZATION-DATA.
```

Notes:

Data-name is the name of an item described in the Working-Storage section of the Data division. The data obtained by the ACCEPT statement is automatically moved into this item.

The data moved into the item will come from the leftmost positions of the input block. Thus, the item may be four bytes long if only the leftmost four characters of the block are desired; or 25 bytes long if the leftmost 25 characters are desired; and so on, to the maximum number of characters.

Data-name must represent either an elementary item described in a level-77 entry, or a record composed of two or more items. In the latter case, data-name would be the name given in the level-01 entry of the record description.

Function To obtain as many as 72 characters of data from the console keyboard. This enables the program to get information from the console operator during the running of the job.

Format ACCEPT data-name FROM CONSOLE

Example

```
ACCEPT CURRENT-DATE FROM CONSOLE.
```

Notes This statement causes a message code generated by the system to be typed on the console printer, followed by the words 'AWAITING REPLY'. Execution of the program is then suspended until the operator uses the keyboard to type the same message code and the information needed by the program.

The operator must, of course, know what information he is required to type on the keyboard. A frequent practice is first to use a DISPLAY UPON CONSOLE statement to type instructions to the operator on the console printer, and then to use the ACCEPT FROM CONSOLE statement to obtain the operator's reply.

All of the Notes for ACCEPT (1) also apply to this statement.

Function

To add the values of two or more items, and to place the sum into one of the items.

Format

```
ADD   {data-name-1      }   [data-name-2      ] ...
      {numeric-literal-1}   [numeric-literal-2]

  TO  data-name-n
```

Example

```
ADD GROSS-EARNINGS TO PLANT-TOTAL.
```

Notes

All data-names must represent elementary numeric items. Implied decimal points are aligned.

The values of all of the items are added, including the value of data-name-n.

The sum replaces the original value of data-name-n. The sum is not edited.

The word GIVING cannot be written in the same statement with the word TO.

Function To add the values of two or more items, and to place the sum into some other item.

Format

```
ADD   {data-name-1      }   {data-name-2      }
      {numeric-literal-1}   {numeric-literal-2}

     [data-name-3      ]
     [numeric-literal-3]  ...   GIVING   data-name-n
```

Example

```
ADD QUANTITY-ON-HAND,
    QUANTITY-ON-ORDER,
    GIVING QUANTITY-AVAILABLE.
```

Notes

All data-names except data-name-n must represent elementary numeric items. Implied decimal points are aligned.

At least two data-names or numeric-literals must precede the word GIVING. The word TO cannot be written in the same statement with the word GIVING.

Data-name-n must represent either an elementary numeric item or an elementary report item. The value of data-name-n is not added.

The sum replaces the original value of data-name-n, and is edited according to that item's picture.

CLOSE

Function To terminate the processing of one or more data files, and to release the storage areas that were used as buffers for those files. When applicable, end-of-file label records (trailer labels) are written, and the file is repositioned at its beginning.

Format CLOSE file-name-1 [file-name-2 ...]

Example

```
    CLOSE SALES-STATISTICS.
```

Notes Any file that was opened must be closed before the run is stopped.

All files may be closed at one time, or each file may be closed as soon as its processing is finished.

Files are not identified as "input" or "output" files.

After a file has been closed, neither that file, nor any record or data item in that file, may be referred to in a procedure; except that the file may be opened again, using an OPEN statement. A common technique is to open a file (say, a magnetic tape file) on which to write the output of a segment of a program; the file is closed at the end of that segment, at which time the tape is rewound; then the file is opened again, this time as an input file for the next segment of the program.

Function To place the value of an arithmetic expression, data item, or numeric literal into a data item.

Format

```
COMPUTE   data-name-1 = { arithmetic-expression }
                        { data-name-2           }
                        { numeric-literal       }
```

Example

```
    COMPUTE NEW-BALANCE = OLD-BALANCE +
        RECEIPTS - ISSUES.
```

Notes The value of whatever is written to the right of the equal sign is placed into the item represented by data-name-1.

Data-name-1 must represent either an elementary numeric item or an elementary report item. The value placed into the item is edited according to the item's picture.

Arithmetic-expressions are discussed on the following page. In addition to the information given there, it should be noted that data-name-1 may be used in the arithmetic expression; for instance, it is possible to write COMPUTE TOTAL = TOTAL + INCREASE - DECREASE. In this example, the original value of TOTAL is used to compute a new value to be placed into TOTAL. When data-name-1 is used in this way, it must represent an elementary numeric item, and there is no editing of the result.

When the data-name-2 or numeric-literal option is used, the result is the same as if a MOVE had been written. For example, COMPUTE RESULT = 100 is equivalent to MOVE 100 TO RESULT.

Data-name-2 must represent an elementary numeric item.

The figurative constant ZERO may be used in place of the numeric-literal 0.

ARITHMETIC EXPRESSIONS

Function

To express an algebraic formula consisting of a series of two or more operands to be added, subtracted, multiplied, divided or exponentiated. The formula may call for any number of these operations, in any sequence. Arithmetic expressions are used in COMPUTE entries and IF entries.

Format

```
{data-name-1       }   ( + )   {data-name-2       }
{numeric-literal-1 }   ( - )   {numeric-literal-2 }
                       ( * )
                       ( / )
                       ( ** )

[ ( + )   {data-name-3       } ]  ...
  ( - )   {numeric-literal-3 }
  ( * )
  ( / )
  ( ** )
```

Example

```
(1 + RATE / BASE) ** (YEARS * BASE) - 1
```

Notes

Data-names must represent elementary numeric items.

Operations are represented by symbols, as follows: + stands for "plus"; - for "minus"; * for "times"; / for "divided by"; and ** for "raised to the power of". (On the program sheet, these symbols must be preceded and followed by spaces.)

Normally, operations are performed in order from left to right, with all exponentiations being performed first, then all multiplications and divisions, and finally all additions and subtractions.

The normal order of operations may be changed by using parentheses to enclose two or more operands and the symbols between them, such as (REGULAR-HOURS + 1.5 * OVERTIME-HOURS). Operations in parentheses are performed prior to operations outside parentheses. When there are parentheses within parentheses, such as ((ON-HAND + ON-ORDER) * UNIT-PRICE), the operations in the innermost set of parentheses are performed first.

Work areas needed to perform calculations are provided by the compiler.

Implied decimal points are aligned where required.

Function

To put as many as 120 characters of data out on the output device which has been designated as the "system logical output device". This output device might be a disk file, tape drive, printer, etc.; sooner or later, though, the data will be printed out. This statement is used for low-volume output, such as exception records, control totals, or messages; it is not used to write files of data.

Format

```
DISPLAY { data-name-1                } [ data-name-2                ] ...
        { literal-1                  }   literal-2
        { figurative-constant-1      }   figurative-constant-2
```

Example

```
DISPLAY ACCOUNT-NUMBER, FINAL-TOTAL.
```

Notes

Data-names can represent either group or elementary items. Literals can be either numeric or non-numeric.

The contents of the items represented by data-names, or the actual characters represented by literals and figurative-constants, will be displayed.

Any number of data-names, literals, and figurative-constants can be written in one statement, but the combined size of the items must not exceed the specified maximum.

When two or more items are displayed, no spaces are left between them. If a space is desired, either the figurative constant SPACE must be used, or a space must be included in a non-numeric literal. For example: DISPLAY CUSTOMER-NUMBER, SPACE, TRANSACTION-AMOUNT, SPACE, EXCEPTION-CODE.

The quotation marks that enclose non-numeric literals are not displayed. If quotation marks are to be part of the output, the figurative constant QUOTE must be used. For example: DISPLAY 'BEGIN' QUOTE 'PHASE 2' QUOTE. The data that would be displayed in this case is BEGIN 'PHASE 2'

Function

To type as many as 72 characters of data on the console printer. This statement is used to type a one-line message to the operator.

Format

```
DISPLAY { data-name-1                } [ data-name-2                ] ...
        { literal-1                  }   literal-2
        { figurative-constant-1      }   figurative-constant-2

    UPON CONSOLE
```

Example

```
    DISPLAY 'GIVE CURRENT DATE'
        UPON CONSOLE.
```

Notes

All of the Notes for DISPLAY (1) also apply to this statement.

Function

To put as many as 72 characters of data out on the output device which has been designated as the "system logical punch device". This output device might be a disk file, tape drive, reader-punch, etc.; sooner or later, however, the data will be punched into a card. This statement is used for low-volume output, such as exception or summary information, not for punching files of data cards.

Format

```
DISPLAY { data-name-1           } [ data-name-2           ] ...
        { literal-1             }   literal-2
        { figurative-constant-1 }   figurative-constant-2 ]

    UPON SYSPUNCH
```

Example

```
    DISPLAY DISCREPANCY-RECORD
        UPON SYSPUNCH.
```

Notes

An 80-character record is produced, with the data to be displayed put into the leftmost positions.

If fewer than 72 characters are to be displayed, the remaining positions through position 72 are filled with spaces.

Positions 73-80 are reserved for the name of the program. The name placed into these positions is the name given in the PROGRAM-ID paragraph of the Identification division.

All of the Notes for DISPLAY (1) also apply to this statement.

Function To divide the value of one item into the value of another item, and to place the quotient into the second item.

Format

```
DIVIDE { data-name-1    } INTO data-name-2
       { numeric-literal}
```

Example

```
    DIVIDE 12 INTO TWELVE-MONTH-TOTAL.
```

Notes

Data-names must represent elementary numeric items.

The value of data-name-1 or numeric-literal is the divisor.

The original value of data-name-2 serves as the dividend, and is replaced by the quotient. The quotient is not edited.

Function To divide the value of one item into the value of another item, and to place the quotient into a third item.

Format

```
DIVIDE {data-name-1      } INTO {data-name-2      }
       {numeric-literal-1}      {numeric-literal-2}

   GIVING data-name-3
```

Example

```
    DIVIDE PURCHASE-QUANTITY INTO
        TOTAL-COST, GIVING UNIT-COST.
```

Notes

Data-name-1 and data-name-2 must represent elementary numeric items. Data-name-3 must represent either an elementary numeric item or an elementary report item.

The value of data-name-1 or numeric-literal-1 is the divisor.

The value of data-name-2 or numeric-literal-2 is the dividend.

The quotient replaces the original value of data-name-3, and is edited according to that item's picture.

Function To cause a branch to a procedure in the program. The normal flow of control is resumed at the beginning of the specified procedure.

Format GO TO procedure-name

Example

```
    GO TO COMPARE-TRANSACTION-NUMBERS.
```

Notes Procedure-name must be either the name of a paragraph or the name of a section in the Procedure division. The word SECTION is not used after a section name in the statement.

It is permissible to branch from some point in a procedure back to the beginning of the same procedure. In other words, procedure-name may be the name of the procedure that the GO TO statement is part of.

Function To cause one or more statements to be acted on only if a certain condition exists.

Format IF test-condition THEN statement-1 [statement-2 ...].

Example

```
IF BALANCE IS NEGATIVE,
    PERFORM REFUND-PROCEDURE.
```

Notes

Test-condition may be a relation test, sign test, condition-name test, class test, or overflow test. Formats for these tests are given on the page following IF (2).

Any number of statements may follow test-condition. These statements are acted on if the condition exists, and are jumped over if the condition does not exist.

Normal flow of control is followed when the statements are acted on. In other words, statement-1 is acted on first, statement-2 second, and so on. This means that if statement-1 is a GO TO or STOP statement (for instance), then statement-2 will never be acted on. It is up to the programmer to make sure that the statements appear in a logical sequence.

An IF statement must be terminated by a period. All statements following test-condition, up to the period, are dependent on the condition.

Function To cause one or more statements to be acted on only if a certain condition exists, and to cause one or more other statements to be acted on only if the condition does not exist.

Format

```
IF test-condition   THEN   statement-1   [statement-2 ...]

  { ELSE      }
  { OTHERWISE }   statement-m   [statement-n ...] .
```

Example

```
    IF AGE < 21,
        ADD 1 TO NUMBER-OF-MINORS;
    ELSE, ADD 1 TO NUMBER-OF-MAJORS.
```

Notes

Statements up to the word ELSE or OTHERWISE are acted on only if the condition exists.

Statements following ELSE or OTHERWISE, up to the period, are acted on only if the condition does not exist.

All of the Notes for IF (1) also apply to this statement.

Function

To compare two data values; to examine the operational sign of a data item; to check for a specific value in an item; to check the class of data in an item; or to determine whether the form in a printer is at end of page. These tests are used in IF entries.

Relation test format

```
{data-name-1                    }
{literal-1                      }
{arithmetic-expression-1        }  IS  [NOT]
{figurative-constant-1          }

{EQUAL TO                       }  {data-name-2                  }
{GREATER THAN                   }  {literal-2                    }
{LESS THAN                      }  {arithmetic-expression-2      }
{=                              }  {figurative-constant-2        }
{>                              }
{<                              }
```

Sign test format

```
{data-name                      }  IS  [NOT]  {POSITIVE}
{arithmetic-expression          }             {NEGATIVE}
                                              {ZERO    }
```

Condition-name test format

```
[NOT]  condition-name
```

Class test format

```
data-name  IS  [NOT]  {NUMERIC   }
                      {ALPHABETIC}
```

Overflow test format

```
[NOT]  overflow-name
```

Notes

In a relation test, all combinations of operands are permitted, except two literals, two figurative constants, or a literal and a figurative constant.

Arithmetic-expressions are discussed on the page following COMPUTE.

Condition-name must be defined in a level-88 entry in the Data division.

Overflow-name must be defined in an APPLY entry in the Environment division.

MOVE

Function

To move data to one or more receiving items. Editing and/or conversion of the data may occur when the data is moved. Editing occurs when data is moved to an elementary report item. Conversion occurs when data is moved to an elementary numeric item whose usage is different (for example, when data from a DISPLAY item is moved to a COMPUTATIONAL-3 item.)

Format

```
MOVE { data-name-1          } TO data-name-2 [data-name-3 ...]
     { literal              }
     { figurative-constant  }
```

Example

```
MOVE ZEROS TO HIGH-SCORE, LOW-SCORE,
     AVERAGE-SCORE.
```

Notes

There may be any number of receiving items, but only one data source. The same data is moved to all receiving items.

The choice of source and receiving items is subject to the following rules:

1. Data may be moved to group items from any source.
2. Data may be moved to elementary alphanumeric items from any source except from an elementary floating-point item.
3. Data may be moved to elementary numeric items or to elementary report items only from an elementary numeric item, a numeric literal, or the figurative constant ZERO (ZEROS, ZEROES).
4. Data may be moved to elementary alphabetic items only from an elementary alphabetic item, an alphabetic literal, an elementary alphanumeric item, a group item, or the figurative constant SPACE (SPACES).

Function To multiply the value of one item by the value of another item, and to place the product into the second item.

Format

```
MULTIPLY  {data-name-1   }  BY  data-name-2
          {numeric-literal}
```

Example

```
MULTIPLY .025 BY TAX-BASE.
```

Notes Data-names must represent elementary numeric items.

The original value of data-name-2 is used as a factor in the computation, and is replaced by the product. The product is not edited.

Function To multiply the value of one item by the value of another item, and to place the product into a third item.

Format

```
MULTIPLY  {data-name-1       }  BY  {data-name-2       }
          {numeric-literal-1 }      {numeric-literal-2 }

    GIVING  data-name-3
```

Example

```
MULTIPLY WAGE-RATE BY
    REGULAR-HOURS-WORKED,
    GIVING REGULAR-PAY.
```

Notes

Data-name-1 and data-name-2 must represent elementary numeric items.

Data-name-3 must represent either an elementary numeric item or an elementary report item.

The product replaces the original value of data-name-3, and is edited according to that item's picture.

Function To enter an explanatory sentence into the source program listing. The sentence is printed with the other entries of the Procedure division.

Format NOTE any words, numbers, or symbols.

Example

```
    NOTE -- CONTROL AREA IS FILLED WITH
         9'S AFTER LAST RECORD OF MASTER
         FILE HAS BEEN READ.
```

Notes The object program is in no way affected by a NOTE.

There are no restrictions on the length or contents of a NOTE sentence. Even reserved words can be used.

A NOTE entry must be terminated by a period.

A NOTE must not be the first sentence of a paragraph, unless the whole paragraph is made up of explanatory sentences. (The format of a NOTE paragraph is explained on the next page.)

Function To enter a paragraph of comments into the source program listing. The paragraph is printed with the other paragraphs of the Procedure division.

Format paragraph-name

NOTE any words, numbers, or symbols .

[any sentence ...]

Example

```
COMMENT-6.
    NOTE THAT JUST THE FIRST 3 DIGITS OF
    THE CLASSIFICATION CODE ARE
    ANALYZED. THIS IS DONE IN ORDER TO
    CHOOSE AMONG 8 SPECIALIZED SUB-
    ROUTINES WHICH ANALYZE THE REMAINING
    DIGITS. IF THE FIRST 3 DIGITS ARE
    ILLEGAL, THE RECORD IS DUMPED OUT ON
    THE CHECKPOINT TAPE AND THE NEXT
    RECORD IS READ.
```

Notes Coding format rules must be observed, the same as for any other paragraph. The paragraph header entry must begin in margin A, and one or more sentences must be written in margin B.

Only the first sentence of the paragraph must start with the word NOTE. The other sentences may start with any word or character.

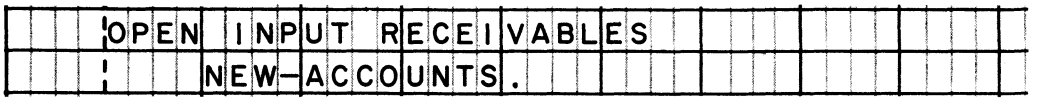
The entire paragraph must be made up of comments. Even if one of the sentences in the paragraph was intended to be a MOVE or ADD statement (for instance), the compiler will treat it as a comment, and disregard it while compiling the object program. Therefore, it is necessary to start a new paragraph for the next procedural statement after a NOTE paragraph.

All of the Notes for NOTE (1) also apply to this format.

Function — To make one or more input data files ready for reading, and to prepare buffers (storage areas) to receive blocks of data. When applicable, header labels (beginning-of-file label records) are checked to make sure that the file is current and that it contains appropriate data for the job. An OPEN statement does not make any records available for processing; a READ statement is required to do that.

Format — OPEN INPUT file-name-1 [file-name-2 ...]

Example



Notes — An input file must be opened to enable READ statements to obtain records for processing.

All files may be opened with one OPEN statement. It is also possible to use a separate OPEN statement for each file.

Each file named in an OPEN statement must be defined in a file description entry in the Data division of the program.

Function To make one or more output files ready for writing, and to prepare buffers (storage areas) to transmit blocks of data. When applicable, the old header labels (beginning-of-file label records) are checked to make sure that it is all right to destroy the old information recorded on the output volume (magnetic tape, magnetic disk, etc.); and new header labels are written. An OPEN statement does not write any data records in the file; a WRITE statement is required to do that.

Format OPEN OUTPUT file-name-1 [file-name-2 ...]

Example

```
OPEN OUTPUT NEW-PAYROLL-FILE.
```

Notes An output file must be opened to enable WRITE statements to release output records.

All of the Notes for OPEN (1) also apply to this statement.

Function — To combine the function of opening one or more input files, with the function of opening one or more output files. These functions are discussed on the preceding two pages.

Format

```
OPEN INPUT    file-name-1    [file-name-2 ...]
     OUTPUT   file-name-m    [file-name-n ...]
```

Example

```
    OPEN INPUT, INVENTORY, BACK-ORDERS;
         OUTPUT, STOCK-STATUS.
```

Notes — All of the Notes for OPEN (1) and (2) also apply to this statement.

Function To cause a branch to a procedure in the program, and when the end of that procedure is reached, to cause control to return to the next statement after the PERFORM statement. In other words, the procedure is treated as a subroutine, and the PERFORM statement in the main routine provides the linkage to and from the subroutine.

Format PERFORM procedure-name

Example

```
    PERFORM NET-PAY-CALCULATION.
```

Notes Procedure-name can be either the name of a paragraph or the name of a section in the Procedure division. The word SECTION is not used after a section name in the statement.

Procedure-name must not be the name of a procedure that the PERFORM statement is part of.

The procedure to be performed can be separate from the main routine. Or, it can be part of the main routine; that is, it can be situated so as to be executed "in line" -- during the normal flow of control from procedure to procedure -- as well as being performed "out of line".

A procedure can be named in more than one PERFORM statement.

A procedure that is performed may itself contain other PERFORM statements.

The instructions that cause control to return to the main routine are generated by the compiler. In a procedure to be performed, the programmer writes only the statements to do the required processing; he does not write a GO TO statement to branch back to the main routine. In fact, a GO TO statement must not be the last statement of a procedure that is to be performed.

Function — To cause a branch to the first procedure in a series of two or more procedures, and when the end of the last procedure in the series is reached, to cause control to return to the next statement after the PERFORM statement.

Format — **PERFORM** procedure-name-1 **THRU** procedure-name-2

Example

```
PERFORM MINOR-TOTAL
    THRU MAJOR-TOTAL.
```

Notes — There may be any number of procedures in the series. Procedure-name-1 is the name of the first procedure, and procedure-name-2 is the name of the last procedure in the series.

The PERFORM statement must not be part of any procedure in the series. However, the procedures in the series may contain other PERFORM statements that name procedures outside the series.

The procedures in the series must actually follow one another in the COBOL program. They must be arranged in the order in which they are to be performed.

All of the Notes for PERFORM (1) also apply to this statement.

READ

Function

To make a data record from a sequential input file available for processing, and to cause one or more statements to be acted on when the end of the file is reached. Data records are made available one at a time from the input blocks. When the end of a volume is reached for a multi-volume file, volumes are switched; for example, at the end of a tape reel, the tape is rewound and the next reel is read. Normally, standard header and trailer labels are checked.

Format

READ file-name AT END imperative-statement-1

[imperative-statement-2 ...] .

Example

```
READ FORECAST-FILE;
    AT END, CLOSE FORECAST-FILE,
            STOP RUN.
```

Notes

This format is used for sequential files, including magnetic tape files, card files, and sequentially-organized files on direct-access devices. A different format is used for non-sequential files.

The file must have been opened before it can be read.

A record remains available until the file is read again or closed.

If the file contains more than one type of record, the next record is made available, regardless of type. (The programmer cannot specify the type of record to be read. Note that the format requires the name of the file, not of a record.) The differing records share the same input area, so the program must determine which type of record is available in the input area at any given time.

The statements that follow the words AT END are acted on when a READ is called for, but the last data record of the file has already been read. Conditional statements such as IF statements are not allowed in the AT END clause. However, a branch may be specified (AT END, GO TO procedure-name) to a procedure that may contain conditional as well as imperative statements.

A READ statement must be terminated by a period. All statements following the words AT END, up to the period, are taken to be end-of-file actions.

Function To terminate execution of the object program.

Format STOP RUN

Example

```
        STOP RUN.
```

Notes Control of the computer is turned over to the operating system control program. In other words, when STOP RUN is acted on, the computer does not halt, but goes on to some other job.

STOP RUN is used at the end of a job, or when an error in the data is so serious that it is impossible to continue the run.

Function To type a message on the console printer, and then to delay the execution of the object program until the operator takes required steps.

Format STOP literal

Example

```
STOP 'HALT 350 -- CONSULT RUN BOOK'.
```

Notes Literal may consist of as many as 72 characters if it is non-numeric, or as many as 18 digits if it is numeric.

A message code generated by the system is typed on the console printer, followed by the literal. Execution of the program is suspended until the operator uses the keyboard to type the same message code. Execution then is resumed at the statement following the STOP statement.

The literal usually consists of brief instructions to the operator. If the instructions require more than 72 characters, a common practice is to make the literal a code number that refers the operator to detailed instructions contained in the run book.

Function

To subtract the values of one or more items from the value of another item, and to place the difference into the latter item.

Format

```
SUBTRACT  {data-name-1      }  [data-name-2      ]  ...
          {numeric-literal-1}  [numeric-literal-2]

    FROM  data-name-n
```

Example

```
    SUBTRACT REFUND FROM AMOUNT-DUE.
```

Notes

All data-names must represent elementary numeric items. Implied decimal points are aligned.

The values of all of the data items named before the word FROM are subtracted from the value of data-name-n.

The difference replaces the original value of data-name-n. The difference is not edited.

Function

To subtract the values of one or more data items from the value of another item, and to place the difference into still another item.

Format

```
SUBTRACT  {data-name-1      }  [data-name-2      ]  ...
          {numeric-literal-1}  [numeric-literal-2]

    FROM  {data-name-m      }  GIVING  data-name-n
          {numeric-literal-m}
```

Example

```
SUBTRACT DEDUCTIONS FROM GROSS-PAY,
    GIVING NET-PAY.
```

Notes

All data-names except data-name-n must represent elementary numeric items. Implied decimal points are aligned.

The values of all of the data items named before the word FROM are subtracted from the value of data-name-m or numeric-literal-m. The calculation is done in a register or in a work area, so the original value of data-name-m or numeric-literal-m remains unchanged after the subtraction.

Data-name-n must represent either an elementary numeric item or an elementary report item. The value of data-name-n is not used in the calculation of the difference.

The difference replaces the original value of data-name-n, and is edited according to that item's picture.

Function To release a data record for a sequential output file. Each output block is filled with the number of records specified in the file description entry in the Data division. When the end of a volume is reached for a multi-volume file, volumes are switched; for example, at the end of a tape file, the tape is rewound and the next reel is written. Normally, standard header and trailer labels are written.

Format **WRITE** record-name

Example

```
WRITE INVENTORY-MASTER.
```

Notes This format is used for sequential files, such as magnetic tape files or sequentially-organized files on direct-access devices. (The same format is used to write non-sequential files; however, its function is different. Different formats, discussed on the following pages, are used for files that are punched or printed.)

The record is written in the file with which its record description is associated in the Data division. Note that the format requires the name of the record, not of a file. The file must have been opened before a record can be written in it.

When a record is released, it is no longer available for processing. Accordingly, all desired processing of the record must be done prior to the WRITE action.

Function To cause a card to be punched in an output card file, and to select a stacker for the card.

Format **WRITE** record-name **AFTER** data-name

Example

```
WRITE SALES-SUMMARY
      AFTER SELECTING-STACKER-4.
```

Notes Record-name is the name of the card record to be punched.

The output record area must contain one more byte than the number of characters to be punched. For the IBM 2540, for instance, the output area must contain 81 bytes, even though data characters are punched from only 80 bytes. The extra byte must be the first byte of the output area; it should be defined in the record description as a FILLER item whose picture is X. The extra byte is required by the compiler, and used for stacker-selection purposes.

Data-name represents an independent item whose value determines which stacker will be selected for the card. The item's picture must be X, and its value must be either V or W. The value V causes the selection of stacker 4 (on the 2540) or stacker 1 (on the IBM 1442); W causes the selection of stacker 8 (2540) or 2 (1442).

All of the Notes for WRITE (1) also apply to this statement.

Function To cause a line to be printed, and to control spacing or skipping.

Format

```
WRITE  record-name  AFTER ADVANCING  {data-name}  LINES
                                      {integer }
```

Example

```
WRITE HEADING-LINE-3
    AFTER ADVANCING 2 LINES.
```

Notes

Record-name is the name of the line to be printed.

The output record area must contain one more byte than the capacity of the printer; for example, it must contain 133 bytes for a 132-position printer. The extra byte must be the first byte of the output area; it should be defined in the record description as a FILLER item whose picture is X. The extra byte is required by the compiler, and used for form-control purposes.

Data-name represents an independent item whose value determines how the forms will be spaced or skipped. The item's picture must be X; its permissible values and their meanings are:

+ (plus sign)	no space (suppress spacing)
space (blank)	single space
0 (zero)	double space
- (hyphen)	triple space
1 through 9	skip to channel 1 through 9, respectively
A, B, C	skip to channel 10, 11, 12, respectively

Integer can be 0, 1, 2, or 3, only. The integer 0 designates a skip to channel 1. The integers 1, 2, and 3 designate single, double, and triple spacing, respectively.

All of the Notes for WRITE (1) also apply to this statement.

STUDENT'S COMMENT FORM

SYSTEM/360 COBOL - Writing Programs in COBOL Reference Handbook
R29-0211-2

Your comments, as well as answers to the following questions, will help us design and administer programmed or self-study courses in a way that better suits your needs. If your answer to a question is "No", or needs further explanation, please use the space provided below.
Comments and suggestions become the property of IBM.

- What is your occupation? _____
- Why did you take this course? _____

	Yes	No
• Did this course, in general, meet your needs?	☐	☐
• Did an IBM employee serve as your advisor?	☐	☐
• Did you receive a completion certificate?	☐	☐
• Did you find the material: Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Well illustrated?	☐	☐
• Did you feel that any particular topics should be added or emphasized?	☐	☐
• Did you feel that any particular topics should not have been included?	☐	☐

- If you found any technical errors, please list them below, giving form number, page, and frame number.

Staple

Fold

Fold

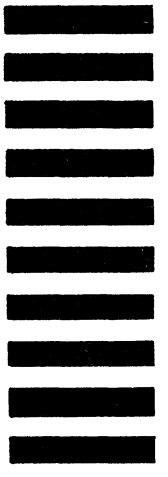
FIRST CLASS
PERMIT NO. 10
ENDICOTT, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY....

IBM Corporation
1701 North St.
Endicott, N.Y. 13760

Attention: DP Education Development, Dept. 617



Fold

Fold

Cut Along Line

Additional Comments:

STUDENT'S COMMENT FORM

SYSTEM/360 COBOL - Writing Programs in COBOL Reference Handbook
R29-0211-2

Your comments, as well as answers to the following questions, will help us design and administer programmed or self-study courses in a way that better suits your needs. If your answer to a question is "No", or needs further explanation, please use the space provided below.
Comments and suggestions become the property of IBM.

- What is your occupation? _____
- Why did you take this course? _____

	Yes	No
• Did this course, in general, meet your needs?	☐	☐
• Did an IBM employee serve as your advisor?	☐	☐
• Did you receive a completion certificate?	☐	☐
• Did you find the material: Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Well illustrated?	☐	☐
• Did you feel that any particular topics should be added or emphasized?	☐	☐
• Did you feel that any particular topics should not have been included?	☐	☐

- If you found any technical errors, please list them below, giving form number, page, and frame number.

Staple

Fold

Fold

FIRST CLASS
PERMIT NO. 10
ENDICOTT, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY....

IBM Corporation
1701 North St.
Endicott, N.Y. 13760

Attention: DP Education Development, Dept. 617

Cut Along Line

Fold

Fold

Additional Comments:

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]