COBOL = Common Business Oriented Language by CODASYL = Conference on Data Systems Language in 1959

Versions:

COBOL-59: original 🡪 -61: revisions 🡪 -68: ANSI standard approved 🡪 -74: revisions 🡪 -85: revisions 🡪 -2002: Object-oriented

HLL used for application programs, but not system software; imperative and procedural; self-documenting, huge data processing

Compiled into machine code after checking syntax to an output file = load module

Program Structure:

PROGRAM – the application program

DIVISIONS – 4 divisions of program

SECTIONS – logical subdivision of program logic

PARAGRAPHS – user- or pre-defined name followed by period

SENTENCES – only in Procedure DIVISION

STATEMENTS – COBOL statements that perform processing

CHARACTERS – cannot be divisible

Each program contains 4 division:

identification – mandatory; used to identify program 🡪 requires `PROGRAM-ID` PARAGRAPH

environment – specify input/output files to program. 2 SECTIONS:

configuration = system info where program is written and executed. 2 PARAGRAPHS:

source computer – system used to compile

object computer – system used to execute

input-output = info of files used by program. 2 PARAGRAPHS:

file control – info on external data sets used

I-O control – info on files used

data – define variables in program. 4 SECTIONS:

file = define record structure of file

working-storage = declare temp variables and file structures used

local-storage = like working-storage, but variables are allocated and initialized every time program starts exec

linkage = describe data names received from external program

procedure – program logic made of STATEMENTS. Last STATEMENT must be either STOP RUN or EXIT PROGRAM

Characters: Serial Number = Characters

1 = A - Z

2 = a - z

3 = 0 - 9

4 = space

5 = plus

6 = minus

7 = \*

8 = /

9 = $

10 = ,

11 = ;

12 = .

13 = “

14 = (

15 = )

16 = >

17 = <

18 = :

19 = ‘

20 = =

Coding Sheet: to comply with compilers, 80 character positions on each line of coding sheet

1 - 6 = Column Number = reserved for line numbers

7 = Indicator = \* for comments, - for continuation, / for form feed

8 – 11 = Area A = All COBOL divisions, sections, paragraphs, and some special entries

12 – 72 = Area B = All COBOL statements

73 – 80 = Identification Area = used as needed

Comments:

Line – can be in any column, compiler treats as documentation

Entry – included in optional paragraphs of Identification Division, written in Area B

Figurative Constants: reserved words for constants; Serial Num = Constant

1 = HIGH-VALUES = 1+ chars at highest position in desc order

2 = LOW-VALUES = 1+ chars have zeros in binary

3 = ZERO/ZEROES = 1+ zeros depending on variable size

4 = SPACES = 1+ spaces

5 = QUOTES = single or double

6 = ALL literal = fills data-item with Literal

Data Types:

Elementary items = cant be divided further. uses level num, data name, picture clause, and value clause (optional)

Group items = consists of one or more elementary items. Uses level num = 01, data name, and value clause (optional)

Data description – defines variables in program. 4 terms: Serial Num = Level Num

Level Number = specify level of data in record to differentiate elementary and group items;

1 = 01 = Record Description Entry

2 = 02 to 49 = Group and Elementary items

3 = 66 = Rename Clause items

4 = 77 = items that cant be sub-divided

5 = 88 = condition name entry

Data Name = user-defined name that do not used reserved words to reference memory locations; can be elementary or group type

Picture (PIC) Clause = defines data type, sign, decimal point position, and length (in bytes);

1 = 9 = Numeric

2 = A = Alphabetic

3 = X = Alphanumeric

4 = V = Implicit Decimal

5 = S = Sign

6 = P = Assumed Decimal

Value Clause = initialize data items

Verbs: used in procedure division for data processing

ACCEPT = get data from operating system (use with FROM)

= get data from user (must be passed through JCL)

DISPLAY = show output of program

INITIALIZE = initialize data item (can’t for data names with RENAME clause)

\* numeric = ZEROES, alphabetic/numeric = SPACES

\* with REPLACING, initialized with given value

MOVE = copy data from source to destination

\* truncated if destination PIC is less, added ZEROS or SPACES if more

\* for group, use with CORRESPONDING/CORR

\* from string, append (x:l), where x = start position and l = length

ADD … TO = add two or more numbers and store in destination operand

\* with GIVING, set destination operand

\* for group, use with CORRESPONDING/CORR

SUBTRACT … FROM = add source numbers and subtract from destination

\* with GIVING, set destination data item without subtracting

\* for group, use with CORRESPONDING/CORR

MULTIPLY … BY = multiply two or more numbers and store in destination

\* with GIVING, set destination data item without multiplying

DIVIDE … INTO = divide source number by destination

\* with REMAINDER, store remainder in another destination

\* with GIVING, replace INTO with BY

COMPUTE = write arithmetic expressions to replace single verb operations

Data Layout: description of each field and its’ values

REDEFINES = define storage with different data description

\* if 1+ data items not used simultaneously, then same storage can be utilized for another

\* Level Num can’t be 66 or 88 and must be the same for redefined and redefining

\* don’t use VALUE with redefining item

\* don’t use with level number 01 in File Section

RENAMES = give different name to existing data item or re-group data names with a new name

\* level num 66 is reserved for renames

\* only possible among same-level num items, don’t use for levels 01 or 77

\* data items with OCCUR clause can’t be renamed

USAGE = specify OS where format data is stored

\* if as group, then all elementary will have same

\* options:

DISPLAY = stored in ASCII format with each character taking 1 byte

COMPUTATIONAL/COMP = stored in binary, items must be integers

COMP-1 = like real/float, single-precision FP number, stored as hex, doesn’t accept PIC, 1 word = 4B

COMP-2 = like long/double, double-precision FP number, stored as hex, doesn’t specify PIC, 2 word = 8B

COMP-3 = packed decimal format, 1 digit occupies 0.5B, sign stored in rightmost 0.5B

COPY = include a copybook (selection of code that defines data structure); used in Working-Storage section

Conditional Statements:

IF = check for condition if true or false; syn. IF [cond] THEN […] ELSE […] ENDIF. (period after last END-IF)

Relation = compares two operands, which can be an identifier, literal, or arithmetic expr

\* syn. [Data Name/Arithmetic Operation] [IS] [NOT] [COMPARATOR] [Data Name/Arithmetic Operation]

\* for non-numeric, if unequal size, appended with spaces until equal

Sign = check sign of numeric operand

\* syn. [Data/Arithmetic] [IS] [NOT] [POSITIVE/NEGATIVE/ZERO] [Data/Arithmetic]

Class = check operand if only alphabetic or numeric, including spaces

\* syn. [Data/Arithmetic] [IS] [NOT] [NUMERIC/ALPHABETIC/ALPHABETIC-LOWER/ALPHABETIC-UPPER] [Data/Arithmetic]

Condition-name = user-defined name that has set of values specified by user, acting like Booleans

\* syn. 88 [Condition-Name] VALUE [IS/ARE] [LITERAL] [THRU LITERAL]

\* uses level num 88 and doesn’t have PIC clause

Negated = given with NOT keyword after IF

Combined = given with AND or OR between conditions

EVALUATE = same as switch statement; syn. EVALUATE [TRUE/FALSE] WHEN [cond] [code] … END-EVALUATE.

Loops:

Perform Thru = execute series of statements/paragraphs

\* in-line – statements in PERFORM executed until END-PERFORM

\* out-of-line – statements executed in PERFORM from PARA1 THRU PARA2

Perform Until = executed until condition is true

\* syn. PERFORM [PARA1] [WITH] [PARA2] [BEFORE/AFTER] UNTIL [cond]

Perform Times = execute paragraph given number of times

\* syn. PERFORM [PARA] [num] TIMES

Perform Varying = execute paragraph until condition becomes true

\* syn. PERFORM [PARA] VARYING [data] FROM [num] BY [num] UNTIL [data = cond]

Go to Statement = change flow of execution in forward direction to exit program.

\* if unconditional, syn. GO TO [PARA]

\* if conditional, syn. GO TO [PARA-1] [PARA-x] DEPENDING ON [x] (if x = 1, do 1st, and so on)

String Handling:

INSPECT = used for type 9/A/X; syn. INSPECT [input];

TALLYING = count string characters; syn. TALLYING [output-data] FOR [[ALL] CHARACTERS/char (s)]

REPLACING = replace string characters; syn. REPLACING ALL [char] BY [char]

STRING = concatenate strings

syn. STRING [str1] DELIMITED BY SPACE [str2] DELIMITED BY SIZE INTO [dest] WITH POINTER [count] ON OVERFLOW DISPLAY [msg1] NOT ON

OVERFLOW DISPLAY [msg2] END-STRING.

UNSTRING = split string to substrings

syn. UNSTRING [str] DELIMITED BY SPACE INTO [dest1, …] WITH POINTER [count] ON OVERFLOW DISPLAY [msg1] NOT ON OVERFLOW DISPLAY

[msg2] END-UNSTRING.

Table Processing: aka Arrays

OCCURS = define table on data names with level num 02 to 49 and not on REDEFINEs

One-dim – syn. 01 WS-TABLE. 05 WS-A PIC A(10) OCCURS 10 TIMES.

Two-dim – syn. 01 WS-TABLE. 05 WS-A OCCURS 10 TIMES. 10 WS-B PIC A(10). 10 WS-C OCCURS 5 TIMES. 15 WS-D PIC X(6).

Subscript = access individual table elements, must be positive number and doesn’t need DATA division declarae since auto-made by OCCURS

One-dim – WS-A(n) two-dim – WS-C(n,n)

Index = access table elements, declared with OCCURS using INDEXED BY

syn. 01 WS-TABLE. 05 WS-A PIC A(10) OCCURS 10 TIMES INDEXED BY I.

SET = initialize, increment, decrement index value; used with SEARCH and SEARCH ALL to locate table elements

syn. SET [index] [TO/UP BY/DOWN BY] [value/data-item]

SEARCH = linear search to find elements in table declared with index, auto-increasing index til element is found

syn. SEARCH [data] AT END DISPLAY [msg] WHEN WS-A(I) = WS-SRCH DISPLAY [msg] END-SEARCH.

SEARCH ALL = binary search to find elements in sorted table, index not needing init,

syn. SEARCH ALL [data] AT END DISPLAY [msg] WHEN WS-A(I) = 93 DISPLAY [msg] END-SEARCH.

File Handling: uses PS (Physical Sequential) and VSAM files

\* file structure declared in FILE section

File – collection of related records

Record – collection of fields used to describe entity 🡪 record size = cumulative size of all fields

Physical – info exists on external device; aka block

Logical – info used by program; COBOL can only handle one record at a time, which is the logical record

Field = indicate data stored about element 🡪 field size = num of characters available; attributes:

Primary key – unique fields used to identify a record (ie. id)

Secondary key – fields in record used to search related data (ie. name)

Descriptor – fields that describe entity (ie. gpa)

\* FILE STATUS variable updated after each operation

OPEN = must be first operation performed to handle file

syn. OPEN “mode” file-name; modes:

INPUT = for existing files; read-only, no other operations allowed

OUTPUT = insert records to file (if sequential, delete existing records then insert new; if indexed/relative, then appended at EOF)

EXTEND = append records to sequential files (if access is random/dynamic, can’t be used)

I-O = read and write records of a file

READ = fetch records from file; at each read verb, only one record can be read in file structure; increments file pointer

(acc = seq) syn. READ file NEXT RECORD INTO ws-file-struct AT END DISPLAY ‘msg’ NOT AT END DISPLAY ws-file-struct END-READ.

NEXT RECORD = optional, specified when indexed sequential file read sequentially

INTO = optional, ws-file-struct defined in Working-Storage to get values from READ

(acc = rand) syn. READ file RECORD INTO ws-file-struct KEY IS rec-key INVALID KEY … NOT INVALID KEY … END-READ.

WRITE = insert records to file; once record written, no longer in record buffer; before insert, move values into record buffer

(org = seq) syn. WRITE record-buffer [FROM ws-file-struct] END-WRITE.

(org = ind/rel) syn. WRITE record-buffer [FROM ws-file-struct] INVALID KEY … NOT INVALID KEY … END-WRITE.

REWRITE = update records; file opened in I-O mode; used after successful READ, overwriting last record read

(org = seq) syn. REWRITE buffer [FROM ws-file-struct] END-REWRITE.

(org = ind/rel) syn. REWRITE buffer [FROM ws-file-struct] INVALID KEY … NOT INVALID KEY … END-REWRITE.

DELETE = remove record from org ind/rel files in I-O mode; deletes last read record

syn. DELETE file RECORD INVALID KEY … NOT INVALID KEY … END-DELETE. ; if acc = rand, specify record key then delete

START = place file pointer at specific record on org = ind/rel and acc = seq/dyn in INPUT/I-O

syn. START file KEY IS [comparator ] rec-key INVALID KEY … NOT INVALID KEY … END-START.

CLOSE = close file; after operation, vars in file struct not available for processing since link between program and file is lost

syn. CLOSE file.

File Organization:

Sequential – read sequentially (all previous first), write at EOF, can’t delete/shorten/lengthen or reorder record after insert,

can be overwritten if new record length is same as old

syn. INPUT-OUTPUT SECTION. FILE CONTROL. SELECT file-name ASSIGN TO dd-name-jcl ORGANIZATION IS SEQUENTIAL

Indexed sequential – can be accessed sequentially or directly (index); 2 parts:

Data file – contains records in sequential scheme

Index file – contains primary key and its address in data file

sorted index is maintained by relating key to record position in file

alternate index can be created to fetch records

syn. INPUT-OUTPUT SECTION. FILE CONTROL. SELECT file-name ASSIGN TO dd-name-jcl ORGANIZATION IS INDEXED RECORD KEY IS primary

ALTERNATE RECORD KEY IS rec-key

Relative – records ordered by relative address; fastest access to records, but missing intermediate records still take space

read sequentially/indexed, accessed with relative address from start of file, inserted with relative key used to calc rel-address

syn. INPUT-OUTPUT SECTION. FILE CONTROL. SELECT file-name ASSIGN TO dd-name-jcl ORGANIZATION IS RELATIVE RELATIVE KEY IS rec-key

File Access Mode:

Sequential – syn. ORGANIZATION IS [SEQUENTIAL/INDEXED/RELATIVE] ACCESS MODE IS SEQUENTIAL

Sequential = records accessed in same order as insert

Indexed = fetch records using record key values

Relative = fetch records using relative record keys

Random – syn. ORGANIZATION IS [INDEXED/RELATIVE] ACCESS MODE IS RANDOM

Indexed = records accessed according to value placed in key field (primary/(1+) alternate)

Relative = fetch records using relative record keys

Dynamic – syn. ORGANIZATION IS [SEQUENTIAL/ RELATIVE] ACCESS MODE IS DYNAMIC

Supports sequential and random; with relative/indexed, can switch between by using NEXT phrase on READ statement

Subroutine: program that can be compiled but not executed independently

Internal – ex. PERFORM statements

External – ex. CALL verb

CALL = transfer control from one program to another; called program needs:

Linkage Section consists of data elems passed in, without VALUE clause, with PIC clause must be compatible

Procedure Division lists vars passed from calling in same order as CALL verb

EXIT-PROGRAM statement

\* by reference, modified var values reflected in calling prog

syn. CALL sub USING var1, varN.

\* by content, modified var values not reflected in calling prog

syn. CALL sub USING BY CONTENT var1, BY CONTENT varN.

\* Static = program compiled with NODYNAM compiler option = loaded into storage at compile time

\* Dynamic = program compiled with DYNAM and NODLL compiler option = loaded into storage at runtime

Sort:

External – SORT utility

Internal – SORT or MERGE verb

Working = hold records while sort in progress (defined in File-Section under SD entry)

Input = file to sort ascending/descending

Output = file with sorted records

SORT = syn. SORT work ON ASCENDING KEY rec-key1 ON DESCENDING KEY rec-key2 USING input GIVING output.

Lifecycle: open file work in I-O, input in INPUT, output in OUTPUT 🡪 transfer input records to work 🡪 sorts SORT-FILE in sequence by rec-key

🡪 transfer sorted work to output 🡪 close input and output 🡪 delete work

MERGE = combine 2+ identically sequenced files;

syn. MERGE work ON ASCENDING KEY rec-key1 ON DESCENDING KEY rec-key2 USING in1, in2 GIVING out.

Lifecycle: like SORT, but opens multiple input files

DB Interface:

Embedded SQL – preprocessed by SQL processor before compile with COBOL as Host Language; COBOL-DB2 apps use COBOL and DB2

– SQL query output directed to predefined set vars, Host Variables, with additional INTO clause in SELECT statements

DB2 App Prog – SQL statements must be in Procedure Division in Area B delimited between EXEC SQL and ENDEXEC., with all tables declared in WorkingStorage Section using INCLUDE statement

Host Variables – receive/insert data from/to table; declared in Working-Storage for all vals passed between prog and DB2

can’t be group items, but can be grouped in host struct; can’t be Renamed or Redsefined; prefix with colon in SQL statements

SQLCA – SQL comm area where DB2 passes feedback of SQL exec to prog; has predefined vars like SQLCODE for error code; val 000 = success

syn. Data Div. Working-Storage Section. EXEC SQL INCLUDE SQLCA END-EXEC.

Cursors – data struct that hold all query results to handle multiple rows selections at a time; defined in Working-Storage or Procedure Div

DECLARE – nonexecutable statement; syn. EXEC SQL DECLARE cur CURSOR FOR SELECT … END-EXEC.

OPEN – need for cursor to be used; syn. EXEC SQL OPEN cur END-EXEC.

CLOSE – release cursor memory; syn. EXEC SQL CLOSE cur END-EXEC.

FETCH – identify cursor and put val in INTO clause. Coded in loop to get one row at a time; syn. EXEC SQL FETCH cur INTO :varN END-EXEC.