COBOL (**CO**mmon **B**usiness **O**riented **L**anguage)

History.

Developed by 1959 by a group called COnference on Data Systems Language (CODASYL). First COBOL compiler was released by December 1959.

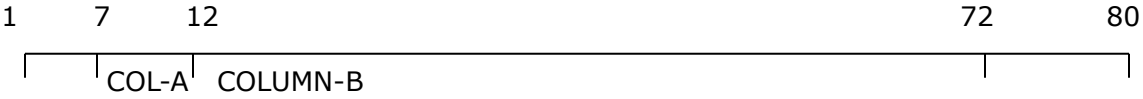
First ANSI approved version – 1968
Modified ANSI approved version – 1974 (OS/VS COBOL)
Modified ANSI approved version – 1985 (VS COBOL 2)

This book is written based on IBM COBOL for OS/390 V2R2.

Speciality.

1. First language developed for commercial application development, which can efficiently handle millions of data.
2. Procedure Oriented Language - Problem is segmented into several tasks. Each task is written as a Paragraph in Procedure Division and executed in a logical sequence as mentioned.
3. English Like language – Easy to learn, code and maintain.

Coding Sheet.



1-6 Page/line numbers – Optional (automatically assigned by compiler)
7 Continuity (-), Comment (*), Starting a new page (/)
Debugging lines (D)
8-11 Column A –Division, Section, Paragraph, 01,77 declarations must begin here.
12.72 Column B –All the other declarations/statements begin here.
73-80 Identification field. It will be ignored by the compiler but visible in the source listing.

Language Structure.

Character	Digits (0-9), Alphabets (A-Z), Space (b), Special Characters (+ - * / ( ) = $; " > < . ,)
Word	One or more characters- User defined or Reserved
Clause	One or more words. It specifies an attribute for an entry
Statement	One or more valid words and clauses
Sentence	One or more statements terminated by a period
Paragraph	One or more sentences.
Section	One or more paragraphs.
Division	One or more sections or paragraphs
Program	Made up of four divisions

Divisions in COBOL.

There are four divisions in a COBOL program and Data division is optional.

1.Identification Division.
2.Environment Division.
3.Data Division.
4.Procedure Division.

Identification Division.

This is the first division and the program is identified here. Paragraph PROGRAM-ID followed by user-defined name is mandatory. All other paragraphs are optional and used for documentation. The length of user-defined name for IBM COBOL is EIGHT.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.       PROGRAM NAME.
AUTHOR.           COMMENT ENTRY.
INSTALLATION.     COMMENT ENTRY.
DATE-WRITTEN.     COMMENT ENTRY.
DATE-COMPILED.    COMMENT ENTRY.
SECURITY.         COMMENT ENTRY.
```

Security does not pertain to the operating system security, but the information that is passed to the user of the program about the security features of the program.

Environment Division.

Only machine dependant division of COBOL program. It supplies information about the hardware or computer equipment to be used on the program. When your program moves from one computer to another computer, the only section that may need to be changed is ENVIRONMENT division.

Configuration Section.

It supplies information concerning the computer on which the program will be compiled (SOURCE-COMPUTER) and executed (OBJECT-COMPUTER). It consists of three paragraphs – SOURCE COMPUTER, OBJECT-COMPUTER and SPECIAL-NAMES. This is OPTIONAL section from COBOL 85.

SOURCE-COMPUTER. IBM-4381 (Computer and model # supplied by manufacturer)
WITH DEBUGGING MODE clause specifies that the debugging lines in the program (statements coded with 'D' in column 7) are compiled.

OBJECT-COMPUTER. IBM-4381 (Usually same as source computer)

SPECIAL-NAMES. This paragraph is used to relate hardware names to user-specified mnemonic names.

1. Substitute character for currency sign. (CURRENCY SIGN IS litearal-1)
2. Comma can be used as decimal point. (DECIMAL-POINT IS COMMA)
3. Default collating sequence can be changed. It will be explained later.
4. New class can be defined using CLASS keyword. (CLASS *DIGIT* is "0" thru "9")

Input-Output Section.

It contains information regarding the files to be used in the program and it consists of two paragraphs FILE-CONTROL & I-O CONTROL.
FILE CONTROL. Files used in the program are identified in this paragraph.
I-O CONTROL. It specifies when check points to be taken and storage areas that are shared by different files.

Data Division.

Data division is used to define the data that need to be accessed by the program. It has three sections.

FILE SECTION	describes the record structure of the files.
WORKING-STORAGE SECTION	is used to for define intermediate variables.
LINKAGE SECTION	is used to access the external data. Ex: Data passed from other programs or from PARM of JCL.

Literals, Constants, Identifier,

1. Literal is a constant and it can be numeric or non-numeric.
2. Numeric literal can hold 18 digits and non-numeric literal can hold 160 characters in it. (COBOL74 supports 120 characters only)
3. Literal stored in a named memory location is called as variable or identifier.
4. Figurative Constant is a COBOL reserved word representing frequently used constants. They are ZERO/ZEROS/ZEROES, QUOTE/QUOTES, SPACE/SPACES, ALL, HIGH-VALUE/HIGH-VALUES, LOW-VALUE/LOW-VALUES.

Example: 01 WS-VAR1 PIC X(04) VALUE 'MUSA'.
'MUSA ' is a non-numeric literal. WS-VAR1 is a identifier or variable.

Declaration of variable

Level# $ Variable $ Picture clause $ Value clause $ Usage Clause $ Sync clause.
FILLER

Level#

It specifies the hierarchy of data within a record. It can take a value from the set of integers between 01-49 or from one of the special level-numbers 66 77 88

01 level.	Specifies the record itself. It may be either a group item or an Elementary item. It must begin in Area A.
02-49 levels.	Specify group or elementary items within a record. Group level items must not have picture clause.
66 level.	Identify the items that contain the RENAMES clause.
77 level.	Identify independent data item.
88 level.	Condition names.

Variable name and Qualifier

Variable name can have 1-30 characters with at least one alphabet in it. Hyphen is the only allowed special character but it cannot be first or last letter of the name. Name should be unique within the record. If two variables with same name are there, then use OF qualifier of high level grouping to refer a variable uniquely.
Ex: MOVE balance *OF* record-1 TO balance *OF* record-2.

FILLER

When the program is not intended to use selected fields in a record structure, define them as FILLER. FILLER items cannot be initialized or used in any operation of the procedure division.

PICTURE Clause

Describes the attributes of variable.

Numeric	9 (Digit), V (Implied decimal point), S (Sign)
Numeric Edited	+ (Plus Sign), - (Minus Sign), CR DB (Credit Debit Sign) . (Period), b (Blank), ‘,’(comma), 0 (Zero), / (Slash) BLANK WHEN ZERO (Insert blank when data value is 0), Z (ZERO suppression), * (ASTERISK), $(Currency Sign)
Non Numeric	A (alphabet), B (Blank insertion Character), X(Alpha numeric), G(DBCS)
Exclusive sets	1. + - CR DB 2. V ‘.’ 3. $ + - Z * (But $ Can appear as first place and * as floating. $***.**)

DBCS (Double Byte Character Set) is used in the applications that support large character sets. 16 bits are used for one character. Ex: Japanese language applications.

Refreshing Basics

Nibble. 4 Bits is one nibble. In packed decimal, each nibble stores one digit.
Byte. 8 Bits is one byte. By default, every character is stored in one byte.
Half word. 16 bits or 2 bytes is one half word. (MVS)
Full word. 32 bits or 4 bytes is one full word. (MVS)
Double word. 64 bits or 8 bytes is one double word. (MVS)

Usage Clause

DISPLAY Default. Number of bytes required equals to the size of the data item.
COMP Binary representation of data item.
PIC clause can contain S and 9 only.
S9(01) – S9(04) Half word.
S9(05) – S9(09) Full word.
S9(10) - S9(18) Double word.
Most significant bit is ON if the number is negative.
COMP-1 Single word floating point item. PIC Clause should not be specified.
COMP-2 Double word floating-point item. PIC Clause should not be specified.
COMP-3 Packed Decimal representation. Two digits are stored in each byte.
Last nibble is for sign. (F for unsigned positive, C for signed positive and D for signed negative)
Formula for Bytes: Integer ((n/2) + 1)) => n is number of 9s.
INDEX It is used for preserve the index value of an array. PIC Clause should not be specified.

VALUE Clause

It is used for initializing data items in the working storage section. Value of item must not exceed picture size. It cannot be specified for the items whose size is variable.

Syntax: VALUE IS literal.
VALUES ARE literal-1 THRU | THROUGH literal-2
VALUES ARE literal-1, literal-2

Literal can be numeric without quotes OR non-numeric within quotes OR figurative constant.

SIGN Clause

Syntax SIGN IS (LEADING) SEPARATE CHARACTER (TRAILING).

It is applicable when the picture string contain 'S'. Default is TRAILING WITH NO SEPARATE CHARACTER. So 'S' doesn't take any space. It is stored along with last digit.

+1=A +2=B +3=C +4=D +5=E +6=F +7=G +8=H +9=I
-0=}, -1= J, -2= K, -3=L, -4=M, -5=N, -6=O, -7=P, -8=Q, -9=R

Number	TRAILING SIGN (Default)	LEADING SIGN	LEADING SEPARATE.
-125	12N	J25	-125
+125	12E	A25	+125

SYNC Clause and Slack Bytes

SYNC clause is specified with COMP, COMP-1 and COMP-2 items. These items are expected to start at half/full/double word boundaries for faster address resolution. SYNC clause does this but it may introduce slack bytes (unused bytes) before the binary item.

```
01 WS-TEST.
    10 WS-VAR1 PIC X(02).
    10 WS-VAR2 PIC S9(6) COMP SYNC.
```

Assumes WS-TEST starts at relative location 0 in the memory, WS-VAR1 occupies zero and first byte. WS-VAR2 is expected to start at second byte. As the comp item in the example needs one word and it is coded with SYNC clause, it will start only at the next word boundary that is 4th byte. So this introduces two slack bytes between WS-VAR1 and WS-VAR2.

REDEFINES

The REDEFINES clause allows you to use different data description entries to describe the same computer storage area. Redefining declaration should immediately follow the redefined item and should be done at the same level. Multiple redefinitions are possible. Size of redefined and redefining need not be the same.

Example:

```
    01 WS-DATE PIC 9(06).
    01 WS-REDEF-DATE REDEFINES WS-DATE.
        05   WS-YEAR   PIC 9(02).
        05   WS-MON    PIC 9(02).
        05   WS-DAY    PIC 9(02).
```

RENAMES

It is used for regrouping of elementary data items in a record. It should be declared at 66 level. It need not immediately follows the data item, which is being renamed. But all RENAMES entries associated with one logical record must immediately follow that record's last data description entry. RENAMES cannot be done for a 01, 77, 88 or another 66 entry.

```
01 WS-REPSONSE.
    05 WS-CHAR143      PIC X(03).
    05 WS-CHAR4        PIC X(04).
    66 ADD-REPSONSE RENAMES WS-CHAR143.
```

CONDITION name

It is identified with special level '88'. A condition name specifies the value that a field can contain and used as abbreviation in condition checking.

```
01 SEX PIC X.
    88 MALE    VALUE '1'
    88 FEMALE VALUE '2' '3'.
```

IF SEX=1 can also be coded as IF MALE in Procedure division.

'SET FEMALE TO TRUE ' moves value 2 to SEX. If multiple values are coded on VALUE clause, the first value will be moved when it is set to true.

JUSTIFIED RIGHT

This clause can be specified with alphanumeric and alphabetic items for right justification. It cannot be used with 66 and 88 level items.

OCCURS Clause

OCCURS Clause is used to allocate physically contiguous memory locations to store the table values and access them with subscript or index. Detail explanation is given in Table Handling section.

LINKAGE SECTION

It is used to access the data that are external to the program. JCL can send maximum 100 characters to a program thru PARM. Linkage section MUST be coded with a half word binary field, prior to actual field. If length field is not coded, the first two bytes of the field coded in the linkage section will be filled with length and so there are chances of 2 bytes data truncation in the actual field.

```
01 LK-DATA.
05 LK-LENGTH         PIC S9(04) COMP.
05 LK-VARIABLE       PIC X(08).
```

LINKAGE section of sub-programs will be explained later.

Procedure Division.

This is the last division and business logic is coded here. It has user-defined sections and paragraphs. Section name should be unique within the program and paragraph name should be unique within the section.

Procedure division statements are broadly classified into following categories.

Statement Type	Meaning
Imperative	Direct the program to take a specific action. Ex: MOVE ADD EXIT GO TO
Conditional	Decide the truth or false of relational condition and based on it, execute different paths. Ex: IF, EVALUATE
Compiler Directive	Directs the compiler to take specific action during compilation. Ex: COPY SKIP EJECT
Explicit Scope terminator	Terminate the scope of conditional and imperative statements. Ex: END-ADD END-IF END-EVALUATE
Implicit Scope terminator	The period at the end of any sentence, terminates the scope of all previous statements not yet terminated.

MOVE Statement

It is used to transfer data between internal storage areas defined in either file section or working storage section.

Syntax:

MOVE identifier1/literal1/figurative-constant TO identifier2 (identifier3)

Multiple move statements can be separated using comma, semicolons, blanks or the keyword THEN.

Numeric move rules:

A numeric or numeric-edited item receives data in such a way that the decimal point is aligned first and then filling of the receiving field takes place.

Unfilled positions are filled with zero. Zero suppression or insertion of editing symbols takes places according to the rules of editing pictures.

If the receiving field width is smaller than sending field then excess digits, to the left and/or to the right of the decimal point are truncated.

Alphanumeric Move Rules:

Alphabetic, alphanumeric or alphanumeric-edited data field receives the data from left to right. Any unfilled field of the receiving filed is filled with spaces.

When the length of receiving field is shorter than that of sending field, then receiving field accepts characters from left to right until it is filled. The unaccomodated characters on the right of the sending field are truncated.

When an alphanumeric field is moved to a numeric or numeric-edited field, the item is moved as if it were in an unsigned numeric integer mode.

CORRESPONDING can be used to transfer data between items of the same names belonging to different group-items by specifying the names of group-items to which they belong.

MOVE CORRESPONDING group-1 TO group-2

Group Move rule

When MOVE statement is used to move information at group level, the movement of data takes place as if both sending and receiving fields are specified as alphanumeric items. This is regardless of the description of the elementary items constituting the group item.

Samples for understanding MOVE statement (MOVE A TO B)

Picture of A	Value of A	Picture of B	Value of B after Move
PIC 99V99	12.35	PIC 999V99	012.35
PIC 99V99	12.35	PIC 9999V9999	0012.3500
PIC 99V999	12.345	PIC 9V99	2.34
PIC9(05)V9(03)	54321.543	PIC 9(03)V9(03)	321.543
PIC 9(04)V9(02)	23.24	PIC ZZZ99.9	23.2
PIC 99V99	00.34	PIC $$$.99	$.34
PIC X(04)	MUSA	XBXBXB	M U S

ARITHMETIC VERBS

All the possible arithmetic operations in COBOL using ADD, SUBTRACT, MULTIPLY and DIVIDE are given below:

Arithmetic Operation	A	B	C	D
ADD A TO B	A	A+B		
ADD A B C TO D	A	B	C	A+B+C+D
ADD A B C GIVING D	A	B	C	A+B+C
ADD A TO B C	A	A+B	A+C	
SUBTRACT A FROM B	A	B-A		
SUBTRACT A B FROM C	A	B	C-(A+B)	
SUBTRACT A B FROM C GIVING D	A	B	C	C-(A+B)
MULTIPLY A BY B	A	A*B		
MULTIPLY A BY B GIVING C	A	B	A*B	
DIVIDE A INTO B	A	B/A		
DIVIDE A INTO B GIVING C	A	B	B/A	
DIVIDE A BY B GIVING C	A	B	A/B	
DIVIDE A INTO B GIVING C REMAINDER D	A	B	Integer (B/A)	Integer remainder

GIVING is used in the following cases:

1.To retain the values of operands participating in the operation.

2.The resultant value of operation exceeds any of the operand size.

ROUNDED option

With ROUNDED option, the computer will always round the result to the PICTURE clause specification of the receiving field. It is usually coded after the field to be rounded. It is prefixed with REMAINDER keyword ONLY in DIVIDE operation.

ADD A B GIVING C ROUNDED.

DIVIDE..ROUNDED REMAINDER

Caution: Don't use for intermediate computation.

ON SIZE ERROR

If A=20 (PIC 9(02)) and B=90 (PIC 9(02)), ADD A TO B will result 10 in B where the expected value in B is 110. ON SIZE ERROR clause is coded to trap such size errors in arithmetic operation.

If this is coded with arithmetic statement, any operation that ended with SIZE error will not be carried out but the statement follows ON SIZE ERROR will be executed.

ADD A TO B ON SIZE ERROR DISPLAY 'ERROR!'.

COMPUTE

Complex arithmetic operations can be carried out using COMPUTE statement. We can use arithmetic symbols than keywords and so it is simple and easy to code. + For ADD, - for SUBTRACT, * for MULTIPLY, / for DIVIDE and ** for exponentiation.

Rule: Left to right – 1.Parentheses
2.Exponentiation
3.Multiplication and Division
4.Addition and Subtraction

Caution: When ROUNDED is coded with COMPUTE, some compiler will do rounding for every arithmetic operation and so the final result would not be precise.

77 A PIC 999 VALUE 10
COMPUTE A ROUNDED = (A+2.95) *10.99
Result: (ROUNDED(ROUNDED(12.95) * ROUNDED(10.99)) =120 or
ROUNDED(142.3205) = 142

So the result can be 120 or 142.Be cautious when using ROUNDED keyword with COMPUTE statement.

All arithmetic operators have their own explicit scope terminators. (END-ADD, END-SUBTRACT, END-MULTIPLY, END-DIVIDE, END-COMPUTE). It is suggested to use them.

CORRESPONDING is available for ADD and SUBTRACT only.

INITIALIZE

VALUE clause is used to initialize the data items in the working storage section whereas INITIALIZE is used to initialize the data items in the procedure division.

INITIALIZE sets the alphabetic, alphanumeric and alphanumeric-edited items to SPACES and numeric and numeric-edited items to ZERO. This can be overridden by REPLACING option of INITIALIZE. FILLER, OCCURS DEPENDING ON items are not affected.

Syntax: INITIALIZE identifier-1
REPLACING (ALPHABETIC/ALPHANUMERIC/ALPHA-NUMERIC-EDITED
NUMERIC/NUMERIC-EDITED)
DATA BY (identifier-2 /Literal-2)

ACCEPT

ACCEPT can transfer data from input device or system information contain in the reserved data items like DATE, TIME, DAY.

ACCEPT WS-VAR1 (FROM DATE/TIME/DAY/OTHER SYSTEM VARS).

If FROM Clause is not coded, then the data is read from terminal. At the time of execution, batch program will ABEND if there is no in-stream data from JCL and there is no FROM clause in the ACCEPT clause.

DATE option returns six digit current date in YYYYMMDD
DAY returns 5 digit current date in YYDDD
TIME returns 8 digit RUN TIME in HHMMSSTT
DAY-OF-WEEK returns single digit whose value can be 1-7 (Monday-Sunday respectively)

DISPLAY

It is used to display data. By default display messages are routed to SYSOUT.
Syntax: DISPLAY identifier1| literal1 (UPON mnemonic name)

STOP RUN, EXIT PROGRAM & GO BACK

STOP RUN is the last executable statement of the main program. It returns control back to OS.

EXIT PROGRAM is the last executable statement of sub-program. It returns control back to main program.

GOBACK can be coded in main program as well as sub-program as the last statement. It just gives the control back from where it received the control.

Collating Sequence

There are two famous Collating Sequence available in computers. IBM and IBM Compatible machine use EBCDIC collating sequence whereas most micro and many mainframe systems use ASCII collating sequence. The result of arithmetic and alphabetic comparison would be same in both collating sequences whereas the same is not true for alphanumeric comparison.

EBCDIC (Ascending Order)	ASCII (Ascending Order)
Special Characters	Special Characters
a-z	0-9
A-Z	A-Z
0-9	a-z

Default collating sequence can be overridden by an entry in OBJECT-COMPUTER and SPECIAL NAMES paragraphs.

1. Code the PROGRAM COLLATING SEQUENCE Clause in the Object computer paragraph. PROGRAM COLLATING SEQUENCE IS alphabet-name

2. Map the alphabet-name in the SPECIAL-NAMES paragraph as follows:
ALPHABET alphabet-name is STANDARD-1 | NATIVE
NATIVE stands for computer's own collating sequence whereas STANDARD-1 stands for ASCII collating sequence.

IF/THEN/ELSE/END-IF

The most famous decision making statement in all language is 'IF'. The syntax of IF statement is given below: IF can be coded without any ELSE statement. THEN is a noise word and it is optional.

If ORs & ANDs are used in the same sentence, ANDs are evaluated first from left to right, followed by ORs. This rule can be overridden by using parentheses.

The permitted relation conditions are =, <, >, <=, >=, <>

CONTINUE is no operation statement. The control is just passed to next STATEMENT. NEXT SENTENCE passes the control to the next SENTENCE. If you forgot the difference between statement and sentence, refer the first page.

It is advised to use END-IF, explicit scope terminator for the IF statements than period, implicit scope terminator.

```
IF condition1 AND condition2 THEN
    Statement-Block-1
ELSE
    IF condition3 THEN
        CONTINUE
    ELSE
        IF condition4 THEN
            Statement-Block-2
        ELSE
            NEXT SENTENCE
        END-IF
    END-IF
END-IF
```

Statement-Block-2 will be executed only when condition 1, 2 and 4 are TRUE and condition 3 is FALSE.

Implied operand: In compound conditions, it is not always necessary to specify both operands for each condition. IF TOTAL=7 or 8 is acceptable. Here TOTAL=8 is implied operation.

SIGN test and CLASS test

SIGN test is used to check the sign of a data item. It can be done as follows –

```
IF identifier is POSITIVE/NEGATIVE/ZERO
```

CLASS test is used to check the content of data item against pre-defined range of values. It can be done as follows -

```
IF identifier is NUMERIC/ALPHABETIC/ALPHABETIC-HIGHER/
                ALPHABETIC-LOWER
```

You can define your own classes in the special names paragraph. We have defined a class DIGIT in our special names paragraph. It can be used in the following way.

```
IF identifier is DIGIT
```

Negated conditions.

Any simple, relational, class, sign test can be negated using NOT.

But it is not always true that NOT NEGATIVE is equal to POSITIVE. (Example ZERO)

EVALUATE

With COBOL85, we use the EVALUATE verb to implement the case structure of other languages. Multiple IF statements can be efficiently and effectively replaced with EVALUATE statement. After the execution of one of the when clauses, the control is automatically come to the next statement after the END-EVALUATE. Any complex condition can be given in the WHEN clause. Break statement is not needed, as it is so in other languages.

General Syntax

```
EVALUATE subject-1 (ALSO subject2..)
    WHEN object-1 (ALSO object2..)
    WHEN object-3 (ALSO object4..)
    WHEN OTHER imperative statement
END--EVALUATE
```

1.Number of Subjects in EVALUATE clause should be equal to number of objects in every WHEN clause.

2.Subject can be variable, expression or the keyword TRUE/ FLASE and respectively objects can be values, TRUE/FALSE or any condition.

3.If none of the WHEN condition is satisfied, then WHEN OTHER path will be executed.

Sample

```
EVALUATE SQLCODE ALSO TRUE
WHEN 100 ALSO A=B imperative statement
WHEN  -305 ALSO (A/C=4) imperative statement
WHEN OTHER imperative statement
END-EVALUATE
```

PERFORM STATEMENTS

PERFORM will be useful when you want to execute a set of statements in multiple places of the program. Write all the statements in one paragraph and invoke it using PERFORM wherever needed. Once the paragraph is executed, the control comes back to next statement following the PERFORM.

1.SIMPLE PERFORM.

```
PERFORM PARA-1.
DISPLAY 'PARA-1 executed'
STOP RUN.
PARA-1.
        Statement1
        Statement2.
```

It executes all the instructions coded in PARA-1 and then transfers the control to the next instruction in sequence.

2.INLINE PERFORM.

When sets of statements are used only in one place then we can group all of them within PERFORM END-PERFORM structure. This is called INLINE PERFORM. This is equal to DO..END structure of other languages.

```
PERFORM
        ADD A TO B
        MULTIPLE B BY C
        DISPLAY 'VALUE OF A+B*C ` C
END-PERFORM
```

3. PERFORM PARA-1 THRU PARA-N.

All the paragraphs between PARA-1 and PARA-N are executed once.

4. PERFORM PARA-1 THRU PARA-N UNTIL condition(s).

The identifiers used in the UNTIL condition(s) must be altered within the paragraph(s) being performed; otherwise the paragraphs will be performed indefinitely. If the condition in the UNTIL clause is met at first time of execution, then named paragraph(s) will not be executed at all.

5. PERFORM PARA-1 THRU PARA-N N TIMES.

N can be literal defined as numeric item in working storage or hard coded constant.

6. PERFORM PARA-1 THRU PARA-N VARYING identifier1
FROM identifier 2 BY identifier3 UNTIL condition(s)

Initialize identifier1 with identifier2 and test the condition(s). If the condition is false execute the statements in PARA-1 thru PARA-N and increment identifier1 BY identifier3 and check the condition(s) again. If the condition is again false, repeat this process till the condition is satisfied.

7.PERFORM PARA-1 WITH TEST BEFORE/AFTER UNTIL condition(s).

With TEST BEFORE, Condition is checked first and if it found false, then PARA-1 is executed and this is the default. (Functions like DO- WHILE)

With TEST AFTER, PARA-1 is executed once and then the condition is checked. (Functions like DO-UNTIL)

Refer Table session for eighth type of PERFORM.

EXIT statement.

COBOL reserved word that performs NOTHING. It is used as a single statement in a paragraph that indicate the end of paragraph(s) execution.
EXIT must be the only statement in a paragraph in COBOL74 whereas it can be used with other statements in COBOL85.

GO TO Usage:

In a structured top-down programming GO TO is not preferable. It offers permanent control transfer to another paragraph and the chances of logic errors is much greater with GO TO than PERFORM. The readability of the program will also be badly affected.

But still GO TO can be used within the paragraphs being performed. i.e. When using the THRU option of PERFORM statement, branches or GO TO statements, are permitted as long as they are within the range of named paragraphs.

```
PERFORM 100-STEP1 THRU STEP-4
..
100-STEP-1.
        ADD A TO B GIVING C.
        IF D = ZERO DISPLAY 'MULTIPLICATION NOT DONE'
                        GO TO 300-STEP3
        END-IF.
200-STEP-2.
        MULTIPLY C BY D.
300-STEP-3.
        DISPLAY 'VALUE OF C:' C.
```

Here GO TO used within the range of PERFORM. This kind of Controlled GO TO is fine with structured programming also!

TABLES

An OCCURS clause is used to indicate the repeated occurrences of items of the same format in a structure. OCCURS clause is not valid for 01, 77, 88 levels. It can be defined as elementary or group item. Initialization of large table occurrences with specific values are usually done using perform loops in procedure division. Simple tables can be initialized in the following way.

```
01 WEEK-ARRAY VALUE 'MONTUEWEDTHUFRISATSUN'.
    05 WS-WEEK-DAYS OCCURS 7 TIMES PIC X(03).
```

Dynamic array is the array whose size is decided during runtime just before the access of first element of the array.

```
01 WS-MONTH-DAY-CAL.
        05 WS-DAYS OCCURS 31 TIMES DEPENDING ON WS-OCCURENCE.
```

IF MONTH = 'FEB' MOVE '28' to WS-OCCURRENCE.

Array Items can be accessed using INDEX or subscript and the difference between them are listed in the table. Relative subscripts and relative indexes are supported only in COBOL85. Literals used in relative subscripting/indexing must be an unsigned integer.

```
ADD WS-SAL(SUB) WS-SAL(SUB + 1) TO WS-SAL(SUB + 2).
```

Sl #	Subscript	Index
1	Working Storage item	Internal Item – No need to declare it.
2	It means occurrence	It means displacement
3	Occurrence, in turn translated to displacement to access elements and so slower than INDEX access.	Faster and efficient.
4	It can be used in any arithmetic operations or for display.	It cannot be used for arithmetic operation or for display purpose.
5	Subscripts can be modified by any arithmetic statement.	INDEX can only be modified with SET, SEARCH and PERFORM statements.

Sometimes, you may face a question like how to randomly access the information in the sequential file of 50 records that contains all the designation and the respective lower and higher salary information.

Obviously, OS does not allow you to randomly access the sequence file. You have to do by yourself and the best way is, load the file into a working storage table in the first section of the program and then access as you wish.

The table look-up can be done in two ways.

-Sequential search.

-Binary search.

Sequential SEARCH

During SERIAL SEARCH, the first entry of the table is searched. If the condition is met, the table look-up is completed. If the condition is not met, then index or subscript is incremented by one and the next entry is searched and the process continues until a match is found or the table has been completely searched.

```
SET indexname-1 TO 1.
SEARCH identifier-1 AT END display 'match not found:'
  WHEN condition-1 imperative statement-1 /NEXT SENTENCE
  WHEN condition-2 imperative statement-2 /NEXT SENTENCE
END-SEARCH
```

Identifier-1 should be OCCURS item and not 01 item.
Condition-1, Condition-2 compares an input field or search argument with a table argument.
Though AT END Clause is optional, it is highly recommended to code that. Because if it is not coded and element looking for is not found, then the control simply comes to the next statement after SEARCH where an invalid table item can be referred and that may lead to incorrect results / abnormal ends.

SET statement Syntax:

```
SET index-name-1 TO/UP BY/DOWN BY integer-1.
```

Binary SEARCH

When the size of the table is large and it is arranged in some sequence – either ascending or descending on search field, then BINARY SEARCH would be the efficient method.

```
SEARCH ALL identifier-1 AT END imperative-statement-1
        WHEN dataname-1 = identifier-2/literal-1/arithmetic expression-1
        AND dataname-2 = identifier-3/literal-2/arithmetic expression-2
END-SEARCH.
```

Identifier-2 and identifier-3 are subscripted items and dataname-1 and dataname-2 are working storage items that are not subscripted.

Compare the item to be searched with the item at the center. If it matches fine, else repeat the process with the left or right half depending on where the item lies.

Sl #	Sequential SEARCH	Binary SEARCH
1	SEARCH	SEARCH ALL
2	Table should have INDEX	Table should have INDEX
3	Table need not be in SORTED order.	Table should be in sorted order of the searching argument. There should be ASCENDING/DESCENDING Clause.
4	Multiple WHEN conditions can be coded.	Only one WHEN condition can be coded.
5.	Any logical comparison is possible.	Only = is possible. Only AND is possible in compound conditions.
6	Index should be set to 1 before using SEARCH	Index need not be set to 1 before SEARCH ALL.
7	Prefer when the table size is small	Prefer when the table size is significantly large.

Multi Dimensional Arrays

COBOL74 supports array of maximum of three dimensions whereas COBOL85 supports up to seven dimensions. The lowest- level OCCURS data-name or an item subordinate to it is used to access an entry in the array or the table.

If we use SEARCH for accessing multi-dimension table, then INDEXED BY must be used on all OCCURS levels. Expanded nested perform is available for processing multi level tables. The syntax of this perform is given below:

```
PERFORM para-1 thru para-n
  VARYING index-1 from 1 BY 1 UNTIL index-1 > size- of- outer-occurs
  AFTER VARYING index-2 from 1 by 1 until index-2 > size of inner occurs.
```

SEARCH example for multi level tables:

```
01 EMP-TABLE.
        05 DEPTNUMBER OCCURS 10 TIMES INDEXED BY I1.
                10 EMP-DETAIL OCCURS 50 TIMES INDEXED BY I2.
                        15 EMP-NUMBER  PIC 9(04).
                        15 EMP-SALARY   PIC 9(05).
77 EMPNUMBER-IN                         PIC 9(04) VALUE '2052'.
PERFORM 100-SEARCH-EMP-SAL VARYING I1 FROM 1 BY 1
    UNTIL  I1 > 10 OR WS-FOUND
100-SEARCH-EMP-SAL.
        SET I2 TO 1.
        SEARCH EMP-DETAIL AT END DISPLAY 'NOT FOUND'     == > Lowest Occurs
           WHEN EMPNUMBER-IN = EMP-NUMBER(I1,I2)
                    DISPLAY 'SALARY IS:' EMP-SALARY(I1,I2)
                     SET WS-FOUND TO TRUE                        == > Search ends
        END-SEARCH.
```

NESTED PROGRAMS, GLOBAL, EXTERNAL

One program may contain other program(s). The contained program(s) may themselves contain yet other program(s). All the contained and containing programs should end with END PROGRAM statement. PGMB is nested a program in the example below:

```
Example:        IDENTIFICATION DIVISION.
                 PROGRAM-ID. PGMA
                 …
                IDENTIFICATION DIVISION.
                PROGRAM-ID. PGMB
                …
                END PROGRAM PGMB.
                …
                END PROGRAM PGMA.
```

If you want access any working storage variable of PGMA in PGMB, then declare them with the clause 'IS GLOBAL' in PGMA. If you want to access any working storage variable of PGMB in PGMA, declare them with the clause 'IS EXTERNAL' in PGMB. Nested Programs are supported only in COBOL85.

If there is a program PGMC inside PGMB, it cannot be called from PGMA unless it's program id is qualified with keyword COMMON.

SORT and MERGE

The programming SORT is called as internal sort whereas the sort in JCL is called external sort. If you want to manipulate the data before feeding to sort, prefer internal sort. In all other cases, external sort is the good choice. Internal sort, in turn invokes the SORT product of your installation. (DFSORT). In the run JCL, allocate at least three sort work files. (SORT-WKnn => nn can be 00-99).

FASTSRT compiler option makes the DFSORT to do all file I-O operation than your COBOL program. It would significantly improve the performance. The result of the SORT can be checked in SORT-RETURN register. If the sort is successful, the value will be 0 else 16.

Syntax:

```
SORT SORTFILE ON ASCENDING /DESCENDING KEY sd-key-1 sd-key2
  USING file1 file2 / INPUT PROCEDURE IS section-1
  GIVING file3       / OUTPUT PROCEDURE is section-2
END-SORT
```

File1, File2 are to-be-sorted input files and File3 is sorted-output file and all of them are defined in FD.SORTFILE is Disk SORT Work file that is defined at SD. It should not be explicitly opened or closed.

INPUT PROCEDURE and USING are mutually exclusive. If USING is used, then file1 and files should not be opened or READ explicitly. If INPUT PROCEDURE is used then File1 and file2 need to be OPENed and READ the records one by one until end of the file and pass the required records to sort-work-file using the command RELEASE. Syntax: RELEASE sort-work-record from input-file-record.

OUTPUT Procedure and GIVING are mutually exclusive. If GIVING is used, then file3 should not be opened or WRITE explicitly. If OUTPUT procedure is used, then File3 should be OPENed and the required records from sort work file should be RETURNed to it. Once AT END is reached for sort-work-file, close the output file. Syntax: RETURN sort-work-file-name AT END imperative statement.

INPUT PROCEDURE Vs OUTPUT PROCEDURE:

Sometimes it would be more efficient to process data before it is sorted, whereas other times it is more efficient to process after it is sorted. If we intend to eliminate more records, then it would be better preprocess them before feeding to SORT. If we want to eliminate all the records having spaces in the key field then it would be efficient if we eliminate them after sorting. Because the records with blank key comes first after sorting.

MERGE

It is same as sort. USING is mandatory. There should be minimum two files in USING.

```
MERGE Sort-work-file ON ASCENDING KEY dataname1 dataname2
  USING file1 file2
  GIVING file3 / OUTPUT PROCEDURE is section-1
END-MERGE
```

Program sort registers (and its equivalent DFSORT parameter/meaning)

SORT-FILE-SIZE (FILSZ), SORT-CORE-SIZE (RESINV), SORT-MSG(MSGDDN)
SORT-MODE-SIZE (SMS=nnnnn)
SORT-RETURN(return-code of sort) and
SORT-CONTROL (Names the file of control card – default is IGZSRTCD)

STRING MANIPULATION

A string refers to a sequence of characters. String manipulation operations include finding a particular character/sub-string in a string, replacing particular character/sub-string in a string, concatenating strings and segmenting strings. All these functions are handled by three verbs INSPECT, STRING and UNSTRING in COBOL. EXAMINE is the obsolete version of INSPECT supported in COBOL74.

INSPECT- FOR COUNTING

It is used to tally the occurrence of a single character or groups of characters in a data field.

```
INSPECT identifier-1 TALLYING identifier-2 FOR
      ALL|LEADING literal-1|identifier-3
     [BEFORE|AFTER INITIAL identifier-4|literal-2]        - Optional.

INSPECT identifier-1 TALLYING identifier-2 FOR
      CHARACTERS
     [BEFORE|AFTER INITIAL identifier-4|literal-2]        - Optional.
```

Main String is identifier-1 and count is stored in identifier-2. Literal-1 or Identifier-3 is a character or group-of-characters you are looking in the main-string. INSPECT further qualifies the search with BEFORE and AFTER of the initial occurrence of identifier-4 or literal-2.

Example:
WS-NAME – 'MUTHU SARAVANA **S**URYA CHANDRA DEVI'

```
INSPECT WS-NAME TALLYING WS-COUNT ALL 'S'
  BEFORE INITIAL 'SARAVANA' AFTER INITIAL 'CHANDRA'
END-INSPECT
```
Result: WS-COUNT contains – 1

INSPECT- FOR REPLACING

It is used to replace the occurrence of a single character or groups of characters in a data field.

```
INSPECT identifier-1 REPLACING
      ALL|LEADING literal-1|identifier-2 BY identifier-3|literal-2
     [BEFORE|AFTER INITIAL identifier-4|literal-2]        - Optional.

INSPECT identifier-1 REPLCING CHARACTERS
      BY identifier-2 BEFORE|AFTER INITIAL identifier-3|literal-1
```

INSPECT-FOR COUNTING AND REPLACING

It is a combination of the above two methods.

INSPECT identifier-1 TALLYING <tallying part > REPLACING <replacing part>

STRING

STRING command is used to concatenate one or more strings.

Syntax:

```
STRING identifier-1 / literal-1, identifier-2/ literal-2
DELIMITED BY (identifier-3/literal-3/SIZE)
INTO identifier-4
END-STRING.
```

```
01 VAR1 PIC X(10) VALUE 'MUTHU    '
01 VAR2 PIC X(10) VALUE 'SARA     '
01 VAR2 PIC X(20).
```

To get display 'MUTHU,SARA'

```
STRING VAR1 DELIMITED BY ' '
       ',' DELIMITED BY SIZE
         VAR2 DELIMITED BY ' '
   INTO VAR3
END-STRING.
```

The receiving field must be an elementary data item with no editing symbols and JUST RIGHT clause.

With STRING statement, specific characters of a string can be replaced whereas MOVE replaces the full string.

```
01 AGE-OUT PIC X(12) VALUE '12 YEARS OLD'.
STRING '18' DELIMITED BY SIZE INTO AGE-OUT.  => 18 YEARS OLD.
```

Reference Modification – equivalent of SUBSTR

'Reference modification' is used to retrieve or overwrite a sub-string of a string. '**:**' is known as reference modification operator.

Syntax: String(Starting-Position**:**Length)

```
MOVE '18' TO AGE-OUT(1:2) does the same as what we did with STRING command.
```

When it is used in array elements, the syntax is

```
Array-element (occurrence) (Starting-Position:Length)
```

UNSTRING

UNSTRING command is used to split one string to many strings.

Syntax:

```
UNSTRING  identifier-1
[DELIMITED BY (ALL/) identifier2/literal1 [,OR (ALL/) (identifier-3/literal-2),..]]
INTO identifier-4 [,DELIMITER IN identifier-5, COUNT IN identifier-6]
 [,identifier-7 [,DELIMITER IN identifier-8, COUNT IN identifier-9]

 01 WS-DATA PIC X(12) VALUE '10/200/300/1'.
 UNSTRING WS-DATA DELIMITED BY '/'
        INTO WS-FLD1 DELIMITER IN WS-D1 COUNT IN WS-C1
             WS-FLD2 DELIMITER IN WS-D2 COUNT IN WS-C2
             WS-FLD3 DELIMITER IN WS-D3 COUNT IN WS-C3
 END-UNSTRING.
```

Result:

```
WS-FLD1 = 10 WS-FLD2 =200 WS-FLD3=300
WS-C1 = 2 WS-C2=3 WS-C3=3 WS-D1 = '/' WS-D2='/' WS-D3 '/'
```

ON OVERFLOW can be coded with STRING and UNSTRING. If there is STRING truncation then the imperative statements followed ON OVERFLOW will be executed.

COPY Statement

A COPY statement is used to bring a series of prewritten COBOL entries that have been stored in library, into a program.

1.Common routines like error routine, date validation routine are coded in a library and bring into the program by COPY.

2. Master files are used in multiple programs. Their layout can be placed in one copybook and be placed wherever the files are used. It promotes program standardization since all the programs share the same layout and the same data names.

This reduces coding and debugging time. Change in layout needs change in copybook only. It is enough if we just recompile the program for making the new copy effective.

Syntax:

COPY copybook-name [(OF/IN) library name]
[REPLACING string-to-be-replaced BY replacing-string]

Copybooks are stored as members in PDS library and during compilation time, they are included into the program. By default, the copybook library is SYSLIB and it can be changed using IN or OF of COPY statement.

Copybooks can be used in the following paragraphs.

SOURCE-COMPUTER, OBJECT-COMPUTER, SPECIAL-NAMES, FILE-CONTROL, IO-CONTROL, FD SECTION, PARAGRAPHS IN PROCEDURE DIVISION.

If the same copybook is used more than once in the program, then there will be "duplicate data declaration" error during compilation, as all the fields are declared twice. In this case, one copybook can be used with REPLACING verb to replace high-level qualifier of the all the variables with another qualifier.

Example: COPY CUSTOMER REPLACING 'CUST1-' BY 'CUST2-'.

Delimiter '= =' should be used for replacing pseudo texts. The replacing option does not alter the prewritten entries in the library; the changes are made to the user's source program only.

CALL statement (Sub-Programs)

When a specific functionality need to be performed in more than one program, it is best to write them separately and call them into each program. Sub Programs can be written in any programming language. They are typically written in a language best suited to the specific task required and thus provide greater flexibility.

Main Program Changes:

CALL statement is used for executing the sub-program from the main program. A sample of CALL statement is given below:

CALL 'PGM2' USING BY REFERENCE WS-VAR1, BY CONTENT WS-VAR2.

PGM2 is called here. WS-VAR1 and WS-VAR2 are working storage items. WS-VAR1 is passed by reference. WS-VAR2 is passed by Content. BY REFERENCE is default in COBOL and need not be coded. BY CONTENT LENGTH phrase permits the length of data item to be passed to a called program.

Sub-Program Changes:

WS-VAR1 and WS-VAR2 are working storage items of main program. As we have already mentioned, the linkage section is used for accessing external elements. As these working storage items are owned by main program, to access them in the sub-program, we need to define them in the linkage section.

```
LINKAGE SECTION.
01 LINKAGE SECTION.
    05 LK-VAR1 PIC 9(04).
    05 LK-VAR2 PIC 9(04).
```

In addition to define them in linkage section, the procedure division should be coded with these data items for address-ability.

```
PROCEDURE DIVISION USING LK-VAR1,LK-VAR2
```

There is a one-one correspondence between passed elements and received elements (Call using, linkage and procedure division using) BY POSITION. This implies that the name of the identifiers in the called and calling program need not be the same (WS-VAR1 & LK-VAR1) but the number of elements and picture clause should be same.

The last statement of your sub-program should be EXIT PROGRAM. This returns the control back to main program. GOBACK can also be coded instead of EXIT PROGRAM but not STOP RUN. EXIT PROGRAM should be the only statement in a paragraph in COBOL74 whereas it can be coded along with other statements in a paragraph in COBOL85.

PROGRAM-ID. <Program-name> IS INITIAL PROGRAM.

If IS INITIAL PROGRAM is coded along with program-id of sub program, then the program will be in initial stage every time it is called (COBOL85 feature). Alternatively CANCEL issued after CALL, will set the sub-program to initial state.

If the sub program is modified then it needs to be recompiled. The need for main program recompilation is decided by the compiler option used for the main program. If the DYNAM compiler is used, then there is no need to recompile the main program. The modified subroutine will be in effect during the run. NODYNAM is default that expects the main program recompilation.

Difference between Pass-by-reference and Pass-by-content

Sl #	Passl By Reference	Pass By Content
1	CALL ‘sub1’ USING BY REFERENCE WS-VAR1	CALL ‘sub1’ USING BY CONTENT WS-VAR1 (BY CONTENT keyword is needed)
2	It is default in COBOL. BY REFERENCE is not needed.	BY CONTENT key word is mandatory to pass an element by value.
3	Address of WS-VAR1 is passed	Value of WS-VAR1 is passed
4	The sub-program modifications on the passed elements are visible in the main program.	The sub-program modifications on the passed elements are local to that sub-program and not visible in the main program.

Difference between Static Call and Dynamic Call

Sl #	STATIC Call	DYNAMIC Call
1	Identified by Call literal. Ex: CALL 'PGM1'.	Identified by Call variable and the variable should be populated at run time. 01 WS-PGM PIC X(08). Move 'PGM1' to WS-PGM CALL WS-PGM
2	Default Compiler option is NODYNAM and so all the literal calls are considered as static calls.	If you want convert the literal calls into DYNAMIC, the program should be compiled with DYNAM option. By default, call variables and any un-resolved calls are considered as dynamic.
3.	If the subprogram undergoes change, sub program and main program need to be recompiled.	If the subprogram undergoes change, recompilation of subprogram is enough.
4	Sub modules are link edited with main module.	Sub modules are picked up during run time from the load library.
5	Size of load module will be large	Size of load module will be less.
6	Fast	Slow compared to Static call.
7	Less flexible.	More flexible.
8	Sub-program will not be in initial stage the next time it is called unless you explicitly use INITIAL or you do a CANCEL after each call.	Program will be in initial state every time it is called.

INTRINSIC FUNCTIONS:

LENGTH	Returns the length of the PIC clause. Used for finding length of group item that spanned across multiple levels.
MAX	Returns the content of the argument that contains the maximum value
MIN	Returns the content of the argument that contains the minimum value
NUMVAL	Returns the numeric value represented by an alphanumeric character string specified in the argument.
NUMVAL-C	Same as NUMVAL but currency and decimal points are ignored during conversion.
CURRENT DATE	Returns 21 Chars alphanumeric value – YYYYMMDDHHMMSSnnnnnn
INTEGER OF DATE	Returns INTEGER equivalent of Gregorian date passed.
INTEGER OF DAY	Returns INTEGER equivalent of Julian date passed.
DATE OF INTEGER	Returns Gregorian date for the integer passed.
DAY OF INTEGER	Returns Julian date for the integer passed.

Note: FUNCTION INTEGER OF DATE (01-01-1601) returns 1.

FILE HANDLING

A data file is collection of relevant records and a record is collection of relevant fields. The file handling in COBOL program involves five steps.

Steps in file-handing

1.Allocation: The files used in the program should be declared in FILE-CONTROL paragraph of environment division. The mapping with JCL DDNAME is done here. The file is allocated to your program by this statement.

2.Definition. The layout of the file and its attributes are defined in the FILE SECTION of DATA DIVISION.

3.Open: Dataset is connected/readied to your program using OPEN statement. The mode of OPEN decides the operation allowed and the initial pointer in the dataset. For example, EXTEND mode allows only write access and the pointer is kept on the end of file to append.

4.Process: Process the file as per requirement, using the I-O statements provided by COBOL. (READ, WRITE, REWRITE and DELETE)

5. Close: After the processing, close the file to disconnect it from the program.

Allocation of file - SELECT Statement
(ENVIRONMENT-> INPUT-OUTPUT-> FILE-CONTROL)

SELECT [OPTIONAL] *FILENAME* ASSIGN to *DDNAME*	=>ALL Files
ORGANIZATION IS SEQUENTIAL/INDEXED/RELATIVE	=>ALL Files
ACCESS IS SEQUNETIAL/RANDOM/DYNAMIC	=>ALL Files
RECORD KEY IS *FILE-KEY1*	=>KSDS
RELATIVE KEY IS *WS-RRN*	=>RRDS
ALTERNARE RECORD KEY IS *FILE-KEY2* WITH DUPLICATES	=>KSDS with
ALTERNARE RECORD KEY IS *FILE-KEY3* WITHOUT DUPLICATES	=>AIX
FILE STATUS IS *WS-FILE-STAT1*	=>ALL Files
[,*WS-FILE-STAT2*]	=>VSAM Files

SELECT Statement- OPTIONAL Clause

This can be coded only for input files. If OPTIONAL is not coded, then the input file is expected to present in JCL. If not, an execution error will occur.

If OPTIONAL is coded, then if the file is not mapped in JCL, it is considered as empty file and the first read results end of file.

The file can also be dynamically allocated instead of static allocation in JCL.

SELECT Statement- ASSIGN TO

FILENAME is the logical name used inside the program and *DDNAME* is the logical name in the JCL, mapped with physical dataset. *DDNAME* can be prefixed with ‘S-‘ to indicate QSAM file, ‘-AS’ to indicate ESDS file and with no prefix to indicate KSDS/RRDS file.

JCL Step executing the program should have a dataset with *DDNAME* as label
//*DDNAME* DD DSN=BPMAIN.EMPLOYEE.DATA,DISP=SHR

SELECT Statement-ORGANIZATION

It can be SEQUENTIAL (PS or VSAM ESDS), INDEXED (VSAM KSDS), RELATIVE (VSAM RRDS). Default is Sequential.

SELECT Statement-ACCESS MODE

SEQUENTIAL.

It is default access mode and it is used to access the records ONLY in sequential order. To read 100th record, first 99 records need to be read and skipped.

RANDOM.

Records can be randomly accessed in the program using the primary/alternate key of indexed file organization or relative record number of relative organization.100th record can directly be read after getting the address of the record from the INDEX part for INDEXED files.100th record can directly be read for RELATIVE files even without any index.

DYNAMIC.

It is mixed access mode where the file can be accessed in random as well as sequential mode in the program.

Example: Reading the details of all the employees between 1000-2000. First randomly access 1000th employee record, then read sequentially till 2000th employee record. START and READ NEXT commands are used for this purpose in the procedure division.

SELECT Statement-RECORD KEY IS

It is primary key of VSAM KSDS file. It should be unique and part of indexed record structure.

SELECT Statement-ALTERNATE RECORD KEY IS

This phrase is used for KSDS files defined with AIX. Add the clause WITH DUPLICATES if the AIX is defined with duplicates.

Referring to VSAM basics, every alternate index record has an associated PATH and the path should be allocated in the JCL that invokes this program.

The DDNAME of the path should be DDNAME of the base cluster suffixed with 1 for the first alternate record clause, suffixed with n for nth ALTERNATE RECORD KEY clause in SELECT clause.

SELECT Statement-FILE STATUS IS *WS-FILE-STAT1,WS-FILE-STAT2*

WS-FILE-STAT1 should be defined as PIC X(02) in working storage section. After every file operation, the file status should be checked for allowable values.

WS-FILE-STAT2 can be coded for VSAM files to get the VSAM return code (2 bytes), VSAM function-code (1 byte) and VSAM feedback code (3 bytes).
This is a 6- byte field in working storage.

RESERVE Clause.

RESERVE clause [RESERVE integer AREA] can be coded in the SELECT statement. The number of buffers to be allocated for the file is coded here.
By default two buffers will be allocated if the clause is not coded. Since similar option is available in JCL, this is not coded in program.

RESERVE 1 AREA allocates one buffer, for the file in the SELECT statement.

Defining the file in FILE SECTION - FD

FD *FILENAME*
RECORDING MODE IS V/VB/F/FB
RECORD CONTAINS M CHARACTERS (TO N CHARACTERS)
BLOCK CONTAINS X CHARACTERS/RECORDS (TO Y CHARACTERS/RECORDS)
LABEL RECORDS ARE OMITTED/STANDARD
DATA RECORD IS *FILE-RECORD*.
01 *FILE-RECORD* PIC X(nnn).

FD-RECORD CONTAINS

It specifies the length of the record in terms of bytes. (It will be RECORD contains m to n CHARACTERS for variable format files)

FD-BLOCK CONTAINS

It specifies the physical record size. It can be mentioned as number of logical records OR number of characters, that is multiple of logical record length. It is suggested to code BLOCK CONTAINS 0 RECORDS so that system will decide the optimum size for the file based on the device used for storing the file. BLOCK CONTAINS clause is treated as comments for VSAM files.

Advantage of Blocking:
1.I-O time is reduced as n numbers of records are read into main memory buffer during an I-O.
2.Inter record gap is removed and the gap exist only between blocks. So memory wastage due to IRG is avoided.

FD-RECORDING MODE IS

It can be F (FIXED) V(VARIABLE) FB(FIXED BLOCK) VB(VARIABLE BLOCKED)
Variable record file identification:

If there is no recording mode/record contains clause, it is still possible to identify variable length records. If there is an OCCURS depending on clause or there are multiple 01 levels and every 01 level is of different size, then the file would be of variable length. Multiple 01 level in File section is an example for implicit redefinition.

FD-LABEL RECORDS Clause

As a general rule, LABEL RECORDS are STANDARD is coded for Disk and Tape files, LABEL RECORDS ARE OMITTED is coded for printer files. In COBOL74, this clause is a mandatory clause whereas COBOL85 made this as optional.

FD-DATA RECORD IS Clause

It is used to name the data record(s) of the file. More than one record can be coded here.

OPEN STATEMENT
Syntax: OPEN OPENMODE *FILENAME*
OPENMODE can be INPUT OUTPUT I-O EXTEND
INPUT - File can be used ONLY-FOR-READ purpose.
OUTPUT - File can be used ONLY-FOR-WRITE purpose.
I-O - File can be used FOR READ, WRITE and REWRITE purpose.
EXTEND - File can be used FOR appending records using WRITE.

CLOSE statement.

The used files are closed using CLOSE statement. If you don’t close the files, the completion of the program closes all the files used in the program.

Syntax: CLOSE *FILENAME*

OPEN and CLOSE for TAPE files - Advanced

If more than one file is stored in a reel of tape, it is called as multi-file volume. When one file is stored in more than one reel of tape, it is called as multi-volume label. One reel is known as one volume. When the end of one volume is reached, automatically the next volume opens. So there is no special control is needed for multi volume files.

```
OPEN INPUT file-1 [WITH NO REWIND | REVERSED]
OPEN OUTPUT file-2 [WITH NO REWIND]
CLOSE file-3 [{REEL|UNIT} [WITH NO REWIND| FOR REMOVAL]
CLOSE file-3 [WITH NO REWIND|LOCK]
```

UNIT and REEL are synonyms.

After opening a TAPE file, the file is positioned at its beginning. When opening the file if the clause REVERSED is coded, then the file can be read in the REVERESE direction. (Provided hardware supports this feature)

When you close the file, the tape is normally rewound. The NO REWIND clause specifies that the TAPE should be left in its current position.

CLOSE statement with REEL option closes the current reel alone. So the next READ will get the first record of next REEL. This will be useful when you want skip all the records in the first reel after n number of records processing.

Since TAPE is sequential device, if you create multiple files in the same TAPE, then before opening the second file, first file should be closed. At any point of time, you can have only one file is active in the program. In addition to this, you have to code MULTIPLE FILE clause in the I-O control paragraph of environment division.

```
MULTIPLE FILE TAPE CONTAINS     OUT-FILE1 POSITION 1
                                OUT-FILE3 POSITION 3.
```

The files OUT-FILE1 and OUT-FILE3 used in the program are part of a same TAPE and they exist in first and third position in the tape. Alternatively, this information can be passed from JCL using LABEL parameter.

READ statement

READ statement is used to read the record from the file.

Syntax: READ *FILENAME* [INTO ws-record] [KEY IS *FILE-KEY1*]

[AT END/INVALID KEY imperative statement1]
[NOT AT END/NOT INVALID KEY imperative statement2]
END-READ

If INTO clause is coded, then the file is directly read into working storage section record. It is preferred as it avoids another move of file-section-record to working-storage-record followed by simple READ. READ-INTO is not preferred for variable size records where the length of the record being read is not known.

KEY IS clause is used while accessing a record randomly using primary/alternate record key.

AT END and NOT AT END are used during sequential READ of the file.

INVALID KEY and NOT INVALID KEY are used during random read of the file. Before accessing the file randomly, the key field should have a value before READ.

WRITE Statement

Write statement is used to write a new record in the file. If the file is opened in EXTEND mode, the record will be appended. If the file is opened in OUTPUT mode, the record will be added at the current position.

Syntax: WRITE *FILE-RECORD* [FROM ws-record]
[INVALID KEY imperative statement1]
END-WRITE

FROM clause avoids the explicit move of working storage record to file section record before WRITE.

REWRITE Statement

REWRITE is used to update an already read record. To update a record in a file, the file should be opened in I-O mode.

Syntax: REWRITE *FILE-RECORD* [FROM ws-record]
[INVALID KEY imperative statement1]
END-REWRITE

START Statement

START is used with dynamic access mode of indexed files. It establishes the current location in the cluster for READ NEXT statement. START itself does not retrieve any record.

Syntax: START *FILENAME*
KEY is EQUAL TO/NOT LESS THAN/GREATER THAN key-name
[INVALID KEY imperative statement1]
END-START.

DELETE Statement

DELETE is used to delete the most recently read record in the file. To delete a record, the file should be opened in I-O mode.

Syntax: DELETE *FILENAME* RECORD
[INVALID KEY imperative statement1]
END-DELETE

File Error – Handling

There are chances for failure of any file I-O processing. The failure of an I-O operation can be accepted or cannot be tolerated. The severity of failure has to be defined in the program design stage.

Let us assume that we don't have any error handling in our program. In this case, for example, if you don't have a specific record in the file, the random read of that record would immediately terminate the program with error 'record not found'.

Error Handling Clauses Provided by COBOL.

The sudden termination can be avoided by handling this error, with INVALID KEY clause of READ. Based on the importance of the record and business rule, we can continue our program with next record or terminate the program properly.
AT END is another error handling clause provided by COBOL. But there is no way to handle all such errors in this way.

Assign file-status and take the responsibility.

The second method is, assigning file-status to the file in the SELECT clause and checks the file status after each and every I-O and ensures that the value of status code is one of the allowable values. If it is not an allowable return code, then abnormally end the program with error statements that would be easier to debug.

But we have to do this checking after each and every I-O operation.
This is MOST PREFERRED ERROR HANDLING METHOD in structured programming.

Declaratives – USE statement

COBOL provides an option to group all the possible errors of specific operation(s) in a place and that will be automatically invoked during the respective operation(s) of any file. This avoids redundant code.

This is done in DECLARATIVE section of the procedure division. DECLARATIVE should be the first section in the procedure division if coded.

```
PROCEDURE DIVISION.
DECLARATIVES.
USE-PROCEDURE SECTION.
        USE AFTER EXCEPTION PROCEDURE ON INPUT
ERROR-PROCEDURE.
        Check the file-status code for validity.
END-DECLARATIVES.
```

Whenever there is an error in the processing of ANY FILE opened in INPUT mode, then the control comes to ERROR-PROCEDURE. The validity of error should be checked in this paragraph and allow or restrict the process down, based on severity of error code.

The complete syntax of USE statements is:

```
        USE AFTER STANDARD ERROR|EXCEPTION PROCEDURE ON
                                    INPUT|OUTPUT|I-O|EXTEND| file-1
```

If INPUT is coded, the following procedure will be executed for every operation involved in any file that is opened in INPUT mode. OUTPUT, I-O and EXTEND have the same meaning but the mode is different.

If file name (file-1) is coded in the USE statement, then all the input-output operation of that specific file will be checked.

ERROR and EXCEPTION are synonyms.

The Procedure written in a DECLARATIVE section should not refer to any non-declarative procedure written after the end procedure and vice-versa.

I-O-CONTROL - SAME AREA AND SAME RECORD AREA

RESERVE clause of SELECT statement specifies the number of buffers to be allocated for a file. SAME AREA allows more than one file to use the same buffer area. This will be very useful when the program must work with a limited memory

space. But the problem is only one file should be open at a time if SAME AREA is coded.

Syntax: SAME AREA FOR file-1 file-2 file-3.

If SAME RECORD AREA is coded, then the buffer is not shared but only the record area is shared. So more than one file can be in open state. We should be careful while filling in the record area of the output file. This may destroy the record read most recently.

Syntax: SAME RECORD AREA FOR file-1 file-2 file-3.

SAME SORT AREA allows more than one sort/merge work files to use the same area. The sort work files are automatically allocated when file is opened and de-allocated when file is closed. As the sort file is automatically opened and closed during a SORT and two sort files cannot be opened at a time, this clause may not be useful.

Syntax: SAME SORT|SORT-MERGE AREA for file-1 file-2.
File-1 or file-2 should be a SD file.

I-O CONTROL- RERUN Clause

RERUN ON rescue FOR EVERY integer RECORDS on file-1

This will cause checkpoint to be taken for every interger-1 records processing of file-1. If the program ABENDED before the complete processing of the file-1, then the program will restart from integer+1ST record instead of first record. The rescue file details should be mentioned outside the program and it varies from installation to installation.

ENTRY statement

ENTRY statement establishes an alternate ENTRY point in a COBOL called sub-program. When a CALL statement naming the alternate entry point is executed in a calling program, control is transferred to the next executable statement following the entry statement. Except when a CALL statement refers to an entry name, the ENTRY statements are ignored at run-time.

Matching Logic

If you have been given two files of similar type, say master and transaction file and you are requested to update the master file with transaction file information for existing records and prepare a report of new transactions and deleted transactions, then you should go for what is called Matching logic. This is also known as co-sequential processing.

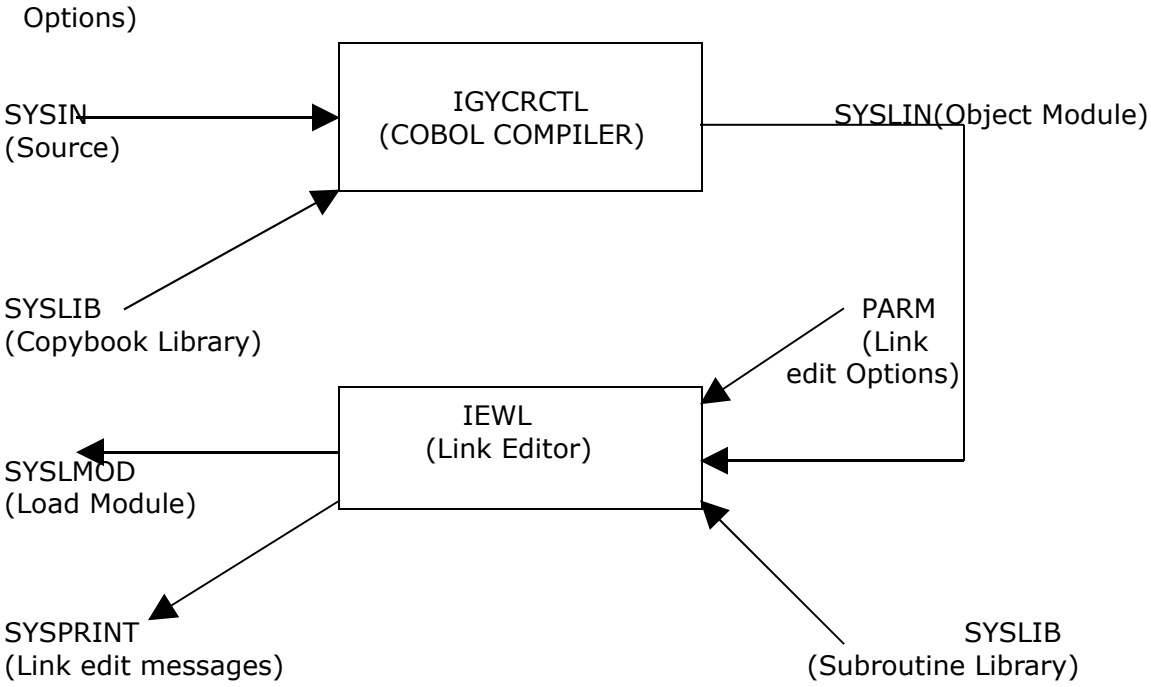
Sort both the files on key and compare the keys. If the keys are matching then update the file. If you find any record that is found in transaction but not in master file, then that is new addition and the reverse is deletion. If the master key is greater than transaction key, then that corresponds to the first case and reverse is the second case.

This can be easily done in JCL using ICETOOL. Refer JCL section.

FILE STATUS CODES

It is a two-byte working storage item. The first byte denotes the general category whereas second byte denotes the particular type of error message under that category.

0		*Successful OPEN/READ/WRITE Operation*
	0	Successful completion
	2	Duplicate key was detected which is allowed as per definition of AIX.
	4	Length of record just READ didn't conform to the fixed length attributes for the file.
	5	Referenced Optional file is not present during OPEN. If open mode is I-O or EXTEND, then file will be created.
	7	Open or Close statement is executed with a phrase that implies a tape file (ex NO REWIND) whereas the file is not in TAPE.
1		*When AT END condition fails*
	0	Sequential READ is attempted on 1.after the end of file is reached 2.optional file that is not present.
	4	Sequential READ was attempted for a relative file and RRN is larger than the maximum that can be stored in the relative key data item.
0		*When INDEX Key fails*
	1	Sequence error exists for sequentially accessed index file.
	2	Attempt was made to write a record that would create a duplicate key.
	3	Record not found.(for keyed random access)
	4	Space not found for WRITE
3		*Permanent Open error*
	5	Opening a non-optional file that was not present.
	7	Open mode is not permitted.
	8	Open issued for a file that was closed previously with lock
	9	File attribute mismatch-Open failed.
4		*Logic error in opening/closing/deleting*
	1	OPEN a opened file.
	2	CLOSE attempted for not opened file.
	3	IO statement before the current REWRITE/DELETE is not successful.
	4	REWRITE attempt with invalid length
	7	READ file which is not opened in INPUT or I-O mode
	8	WRITE file which is not opened in I-O OUPUT or EXTEND mode
	9	DELETE or REWRITE file which is not opened in I-O mode.
9		*Implementation defined*
	1	VSAM password error
	2	Logic error
	3	VSAM resource unavailable
	6	No DD statement specified for VSAM file.
	7	File integrity verified for VSAM file.



COMPILATION JCL:

```
//SMSXL86B JOB ,'COMPILATION JCL', MSGCLASS=Q,MSGLEVEL=(1,1),CLASS=C
//COMPILE1  EXEC  PGM=IGYCRCTL, PARM='XREF,APO,ADV,MAP,LIST),REGION=0M
//STEPLIB     DD  DSN=SYS1.COB2LIB,DISP=SHR
//SYSIN       DD  DSN=SMSXL86.TEST.COBOL(SAMPGM01),DISP=SHR
//SYSLIB      DD  DSN=SMSXL86.COPYLIB,DISP=SHR
//SYSPRINT    DD  SYSOUT=*
//SYSLIN      DD  DSN=&&LOADSET, DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200),
//             DISP=(NEW,PASS),UNIT=SYSDA,SPACE=(CYL,(5,10),RLSE),
//SYSUT1      DD UNIT=&SYSDA,SPACE=(CYL,(1,10))  => Code SYSUT2 to UT7
//LINKEDT1    EXEC  PGM=IEWL,COND=(4,LT)
//SYSLIN      DD  DSN=&&LOADSET, DISP=(OLD,DELETE)
//SYSLMOD     DD  DSN=&&GOSET(SAMPGM01),DISP=(NEW,PASS),UNIT=SYSDA
//             SPACE=(CYL,1,1,1))
//SYSLIB      DD  DSN=SMSXL86.LOADLIB,DISP=SHR
//SYSUT1      DD  UNIT=SYSDA,SPACE=(CYL,(1,10))
//SYSPRINT    DD SYSOUT=*

//*** EXECUTE THE PROGRAM ***
//EXECUTE1  EXEC  PGM=*.LINKEDT1.SYSLMOD,COND=(4,LT),REGION=0M
//STEPLIB     DD  DSN=SMSXL86.LOADLIB,DISP=SHR
//             DD  DSN=SYS1.SCEERUN,DISP=SHR
//SYSOUT      DD  SYSOUT=*
//SYSPRINT    DD  SYSOUT=*
```

Compiler Options

The default options that were set up when your compiler was installed are in effect for your program unless you override them with other options. To check the default compiler options of your installation, do a compile and check in the compilation listing.

Ways of overriding the default options

1.Compiler options can be passed to COBOL Compiler Program (IGYCRCTL) through the PARM in JCL.
2.PROCESS or CBL statement with compiler options, can be placed before the identification division.
3.If the organization uses any third party product or its own utility then these options can be coded in the pre-defined line of the utility panel.

Precedence of Compiler Options

1. (Highest precedence). Installation defaults, fixed by the installation.
2. Options coded on PROCESS /CBL statement
3. Options coded on JCL PARM parameters
4. (Lowest Precedence). Installation defaults, but not fixed.

The complete list of compiler option is in the table:

Aspect	Compiler Option
Source Language	APOST, CMPR2, CURRENCY, DBCS, LIB, NUMBER, QUOTE, SEQUENCE, WORD
Date Processing	DATEPROC, INTDATE, YEARWINDOW
Maps and Listing	LANGUAGE, LINECOUNT, LIST, MAP, OFFSET, SOURCE, SPACE, TERMINAL, VBREF, XREF
Object Deck generation	COMPILE, DECK, NAME, OBJECT, PGMNAME
Object Code Control	ADV, AWO, DLL, EXPORTALL, FASTSRT, OPTIMIZE, NUMPROC, OUTDD, TRUNC, ZWB
Debugging	DUMP, FLAG, FLAGMIG, FLAGSTD, SSRANGE, TYPECHK
Other	ADATA, ANALYZE, EXIT, IDLGEN

ADV: It is meaningful if your program has any printer files with WRITE..ADVANCING keyword. The compiler adds one byte prefix to the original LRECL of printer files for printing control purpose. If you are manually populating printing control character in the program, then you can compile your program with NOADV.

DYNAM: Use DYNAM to cause separately compiled programs invoked through the CALL *literal* statement to be loaded dynamically at run time. DYNAM causes dynamic loads (for CALL) and deletes (for CANCEL) of separately compiled programs at object time. Any CALL *identifier* statements that cannot be resolved in your program are also treated as dynamic calls. When you specify DYNAM, RESIDENT is also put into effect.

LIST/OFFSET: LIST and OFFSET are mutually exclusive. If you use both, LIST will be ignored. LIST is used to produce listing a listing of the assembler language expansion of your code. OFFSET is used to produce a condensed Procedure Division listing. With OFFSET, the procedure portion of the listing will contain line numbers, statement references, and the location of the first instruction generated for each

statement. These options are useful for solving system ABENDS. Refer JCL session for more details.

MAP: Use MAP to produce a listing of the items you defined in the Data Division.

SSRANGE: If the program is compiled with SSRANGE option, then any attempt to refer an area outside the region of the table will abnormally terminate with protection exception, usually S0C4.It also avoids any meaningless operation on reference modification like negative number in the starting position of reference modification expression. If the program is compiled with NOSSRANGE, then the program may proceed further with junk or irrelevant data. So usually the programs are compiled with SSRANGE during development and testing.

RENT: A program compiled as RENT is generated as a reentrant object module. CICS programs should be compiled with RENT option to share the same copy of the program by multiple transactions (Multithreading)

RESIDENT: Use the RESIDENT option to request the COBOL Library Management Feature. (The COBOL Library Management Feature causes most COBOL library routines to be located dynamically at run time, instead of being link-edited with the COBOL program.).CICS Programs should be compiled with RESIENT option.

XREF: Use XREF to get a sorted cross-reference listing. EBCDIC data-names and procedure-names will be listed in alphanumeric order. It also includes listing, where all the data-names that are referenced within your program and the line number where they are defined. This is useful for identifying the fields that are defined but not used anywhere after the development of new program.

TSO Commands from COBOL program

```
CBL APOST,NODECK,OBJECT,BUF(10000),DYNAM         => Compiler option override
     ***************************************************************
     *   FUNCTION = This sample program demonstrates how to invoke    *
     *              TSO commands from a COBOL program using           *
     *              standard TSO services as documented in the        *
```

```
*              TSO/E Programming Services manual.                *
*****************************************************************
 Identification Division.
 Program-ID. SMSTSOEV.

 Data Division.
  Working-Storage Section.
   01 Filler.
     05 ws-dummy        Pic s9(8) Comp.
     05 ws-return-code  Pic s9(8) Comp.
     05 ws-reason-code  Pic s9(8) Comp.
     05 ws-info-code    Pic s9(8) Comp.
     05 ws-cppl-address Pic s9(8) Comp.
     05 ws-flags        Pic X(4) Value X'00010001'.
     05 ws-buffer       Pic X(256).
     05 ws-length       Pic s9(8) Comp Value 256.

 Procedure Division.
*---------------------------------------------------------------*
*          Call IKJTSOEV to create the TSO/E environment         *
*---------------------------------------------------------------*
     CALL 'IKJTSOEV' Using ws-dummy,ws-return-code,ws-reason-code,
                           ws-info-code,ws-cppl-address.
     IF ws-return-code > zero
       DISPLAY 'IKJTSOEV Failed, Return-code=' ws-return-code
                               ' Reason-code=' ws-reason-code
                               'Info-code='    ws-info-code
       MOVE ws-return-code to Return-code
       STOP RUN.
*---------------------------------------------------------------*
*          Build the TSO/E command in ws-buffer                 *
*---------------------------------------------------------------*

     MOVE 'ALLOCATE DD(SYSPUNCH) SYSOUT HOLD' to ws-buffer.

*---------------------------------------------------------------*
*   Call the TSO/E Service Routine to execute the TSO/E command  *
*---------------------------------------------------------------*
     CALL 'IKJEFTSR' Using ws-flags,ws-buffer,ws-length
                           ws-return-code,ws-reason-code,ws-dummy.
     IF ws-return-code > zero
       DISPLAY 'IKJEFTSR Failed, Return-code=' ws-return-code
                               ' Reason-code=' ws-reason-code
       MOVE ws-return-code to Return-code
       STOP RUN.

*---------------------------------------------------------------*
*          Check that the ALLOCATE command worked               *
*---------------------------------------------------------------*
     DISPLAY 'ALLOCATE Worked ! ' Upon Syspunch.

     STOP RUN.
```

Interview Questions(IQ):

* Says importance and possibility of the question in an interview.

1.Difference between Next Sentence and Continue ***

2.Comp, Comp-1, Comp-2 and Comp-3 difference and how many bytes occupied by each. Should know how to read COMP-3 data. *****
3.Identifying and making Static and Dynamic call *****
4.Binary and Sequential search and when you prefer what? *****
5.What are the various ways of passing data from JCL to Program and how to receive them in Program? *****
6.Difference between COBOL74 (OS/VS COBOL) and COBOL85 (VS COBOL2) ****
7.Subscript and Index difference and when you prefer what? ****
8.Reference modification. ****
9.Compiler and Link edit option – SSRANGE MAP LIST OFFSET RENT RESIDENT DYNAM and AMODE/RMODE ***
10.How to make a call by content in COBOL? ***
11.How do you set return code from the program? ***
12.Case structure, Sub-string, Do while, Do Until, BLL equivalent in COBOL ***
13.Difference between section and paragraph *****
14.Can occurs be coded in 01 level? *****
15.Are multiple 01 levels supported in file section? **
16.Various ways of overriding default compiler options **
17.Sort algorithms **
18.How to get the actual length of alphanumeric item? **
19.What is UT-S means with respect to SELECT statement? *
20.Can I rewrite a sequential file in TAPE? *
21.COMP-3 items are always better than COMP in terms of memory. Yes/No **
22.Which VSAM type is fastest? Relative key is part of file section? **
23.How to create a report in COBOL program? ***
24.How to submit a JCL from COBOL program? ****
25.What is SYNC Clause? **
26.What is in line PERFORM and when will you use it? ****
27.What is INSPECT statement? ***
28.To use SEARCH ALL, the table should be in sorted order. I am loading the table from one of the PDS members. The PDS member data is not in sorted order. How will I load the table in SORTED order? You should not sort in JCL. **
29.What is the purpose of USE statement? *
30.What are SAME AREA and SAME RECORD AREA? *
31. Is dynamic allocation possible in COBOL? If yes, How? *
32. What is the difference between ON SIZE ERROR and ON OVERFLOW? *
33.How to swap two variables without third variable? *
34.What is limit of linkage section? *

Answers for selected questions:

What is the limit of working storage and linkage section limit? (IQ 34)

Working storage and Linkage section limit of COBOL85 is 128MB (COBOL74-1MB)

77,01-49 level item limit in COBOL85 is 16MB (COBOL74-1MB)

How to swap the values of two variables without an intermediate variable?(IQ 33)

Let the variables be A and B

```
Way 1:COMPUTE A = A+B        Way 2:    COMPUTE A=A*B
              COMPUTE B = A-B                    COMPUTE B=A/B
              COMPUTE A = A-B                    COMPUTE A=A/B
```

I have retrieved a value from DB2 VARCHAR column. (Ex: WS-VAR = ‘muthu$sara$‘ $ is 1-n spaces.) How to get the length of the WS-VAR in COBOL program? I should not count right hand spaces. (IQ 20)

LENGTH function counts space also as a character. So we cannot use that function for our purpose. INSPECT is also not useful as the string may contain 1- n spaces in between and that needs to be counted. So the logic would be ” Read from right until you read first noon-space character”.

```
PERFORM VARYING WS-SUB-NAME FROM
        LENGTH OF WS-VAR BY -1
        UNTIL END-FOUND OR WS-SUB-NAME = 0
  IF WS-NAME-CHK(WS-SUB-NAME:1) NOT EQUAL TO SPACE
     MOVE 'Y' TO WS-END-OF-FIELD
     DISPLAY 'LENGTH ' WS-SUB-NAME
  END-IF
END-PERFORM
```

How to pass user return code and user ABEND from the COBOL program to the JCL?

RETURN-CODE is a special register and its content is moved to register15 when the control is given back to OS. So move the return code to this register in the program.

Ex: MOVE 1000 to RETURN-CODE.

This sets return code as 1000 for the step that executes this program.

For ABEND, you should call your installation specific assembler routine or ILBOABN0 with the ABEND code you want.

CALL ‘ILBOABN0’ USING WS-AB-CODE.

WS-ABEND-CODE is the variable that has ABEND-CODE. It is a half word binary.

What should be the LRECL of printer files?

Use 133 character records in your program and set the print control character yourself. In this case your JCL would have RECFM=FB, LRECL=133

Use 132 character records in the program and have WRITE ....ADVANCING put in the print control. You need the compiler option ADV for this and the JCL would have RECFM=FBA,LRECL=133.....

What are the sort algorithms? (IQ 17 and 28)

Bubble Sort: Consecutive elements are compared and keys of two elements are not in proper order, they are swapped. In the first pass, the key with largest value will be moved to the last position and n-1 passes needed to sort the whole table.
In between, if any pass results no interchange it implies that the table is in sorted order.

Array: 1 20 9 50 8
First Pass: (Maximum 4 comparisons for 5 elements)
1, 20->no change, 20 & 9 -> 20 is great so swap (1 9 20 50),
20 & 50 -> no change, 50 & 8 -> 50 is great, so swap. (1 9 20 8 50)
Second Pass: (1 9 20 8 50) - (Maximum 3 comparison for 5 elements)
1 & 9-> no change, 9 & 20 -> no change, 20 & 8 -> 20 is great so swap
(1 9 8 20 50)
Third Pass: (1 9 8 20 50) – (Maximum 2 comparisons for 5 elements)
1 & 9 -> no change, 9 & 8-> change (1 8 9 20 50)
Fourth Pass: (1 8 9 20 50) – (Maximum 1 comparison for 5 elements)
1 & 9 -> no change
Note: You can come out of sort when you find 'no change' in all the comparisons of a pass.

Shuttle Sort: In the first pass only first two elements are compared and sorted and in the second pass, third element is compared with two and one and it is placed in the right position. In the ith pass, it assumes that I elements are in already sorted order, proceeds to sort the first (I+1) elements by comparing I+1 th element with I, and I with I-1 and so on until top of the table is reached or no-exchange in a comparison.

Array: 1 20 9 50 8
First Pass: Two elements (1 20) - Maximum 1 comparison
1, 20->no change
Second Pass: Three elements (1 20 9) – Maximum 2 comparisons
9 & 20 -> change (1 9 20) 9 & 1 -> no change
Third Pass: Four elements (1 9 20 50) – Maximum 3 comparisons
50 & 20 -> no change and stop (no need for any other comparison)
Fourth Pass: Five elements (1 9 20 50 8) – Maximum 4 comparisons
8 & 50 -> change (1 9 20 8 50) , 8 & 20-> Change (1 9 8 20 50)
8 & 9 -> Change (1 8 9 20 50) , 8 & 1 -> no change and stop.
Note: You can come out of pass if you find one 'no change'

Shuttle sort is better than bubble sort for sorting arrays with more than 10 elements.

<u>COMP-3 items are always better than COMP with respect to memory usage (IQ 21)?</u>
No. COMP items occupy less space than COMP-3 items at boundaries.
PIC S9(04) COMP occupies 2 bytes whereas PIC S9(04) COMP-3 occupies 3 bytes.
PIC S9(09) COMP occupies 4 bytes whereas PIC S9(09) COMP-3 occupies 5 bytes.
PIC S9(18) COMP occupies 8 bytes whereas PIC S9(18) COMP-3 occupies 10 bytes.

<u>I have a KSDS Students file with 4 bytes key. First two-bytes contain class number and next two-bytes contain student number. I want to read all the students in class '02'. How will you do that?</u>

Allocate the file with dynamic access mode. Move ‘02’ to first two-bytes of the key and low-values to next two-bytes of the key. You can do these moves by reference modification operator or de-grouping the four-byte field into two two-byte fields in the file section.

Issue the START command with KEY IS GREATER THAN clause. Start reading the file in loop until first two-bytes is not equal to 2.