Call Statement, Value Generation, and the Link Parameter in COBOL

1. Call Statement: In COBOL, the `CALL` statement is used to invoke a subprogram from a main program. It allows for modular programming by enabling code reuse and separation of concerns. When a CALL is made, control is passed to the subprogram, and once the subprogram completes, control returns to the calling program.

2. Value Generation: This involves the process of passing data to subprograms. COBOL supports different ways of passing parameters, such as by reference or by value. By generating and passing values correctly, programs ensure that the subprograms operate on the intended data without side effects.

3. Link Parameter: The link parameter in COBOL specifies how data is communicated between the calling program and the called subprogram. It controls the passing of data, either by reference (sharing memory locations) or by value (copying data), affecting the performance and behaviour of the application.

Call Statement, Value Generation, and the Link Parameter in COBOL

1. Call Statement:

The CALL statement in COBOL is used to execute a subprogram dynamically at runtime. It allows the main program (caller) to interact with external routines, facilitating modularity and code reusability. The syntax typically looks like CALL program-name USING parameters. There are two types of calls:

- Static Call: The subprogram is linked at compile time, offering better performance but less flexibility.
- Dynamic Call: The subprogram is linked at runtime, providing flexibility to replace subprograms without recompiling the main program.

Example:

CALL 'SUBPROGRAM' USING PARAM-1, PARAM-2.

2. Value Generation:

COBOL uses different methods for passing parameters between the caller and the subprogram:

- BY REFERENCE: The called subprogram receives a reference to the caller’s data items, meaning it can directly modify the caller’s variables. This is the default method.
- BY CONTENT: A copy of the data is passed to the subprogram, preventing it from altering the caller’s variables.
- BY VALUE: Data is passed by value, usually for numeric or index variables, meaning the subprogram receives a copy and cannot modify the original data.

Example:

CALL 'SUBPROGRAM' USING BY REFERENCE PARAM-1, BY CONTENT PARAM-2.

3. Link Parameter:

The link parameter governs how data is exchanged between the caller and subprogram. The choices between `BY REFERENCE`, `BY CONTENT`, and `BY VALUE` affect memory usage and performance:

- BY REFERENCE: Faster but riskier as subprograms can alter the caller's data.
- BY CONTENT: Safer for protecting data integrity since subprograms work with copies.
- BY VALUE: Ideal for passing small data items or constants, with subprograms receiving only the value.

Calling and Called Programs:

Calling Program: The program that initiates the call to another program.
Called Program: The program that is invoked by the calling program. It receives control and possibly some data from the calling program.

Linkage Section:
The Linkage Section is a part of the called program's Data Division. It defines variables that represent data passed from the calling program. These variables don't occupy memory in the called program itself; they refer to memory locations in the calling program.
The linkage section essentially creates a "link" between the calling program and the called program, allowing data to be passed back and forth.

Procedure Division USING Clause:
In the called program, the Procedure Division begins with a USING clause that specifies the linkage parameters. This clause tells the COBOL compiler which variables (defined in the Linkage Section) will receive data from the calling program.

Data Flow:
When the calling program executes a CALL statement, it specifies the data to pass to the called program. The data is mapped to the variables defined in the Linkage Section of the called program.
The called program can then use, modify, and possibly return this data before passing control back to the calling program.

COBOL Sample Code: Call Statement, Value Generation, and Link Parameter

Main Program (MAINPROG)

This main program calls a subprogram `SUBPROG`, passing parameters by reference, by content, and by value.

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. MAINPROG.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  WS-NUMBER        PIC 9(4) VALUE 1000.
       01  WS-NAME          PIC A(20) VALUE 'MAIN PROGRAM'.
       01  WS-RESULT        PIC 9(4).
       01  WS-INPUT-VALUE   PIC 9(4) VALUE 2000.

       PROCEDURE DIVISION.
       MAIN-LOGIC.
           DISPLAY 'Calling SUBPROG with parameters...'.

           * Calling SUBPROG with parameters
           CALL 'SUBPROG' USING BY REFERENCE WS-NUMBER
                        BY CONTENT WS-NAME
                        BY VALUE WS-INPUT-VALUE
                        BY REFERENCE WS-RESULT.

           DISPLAY 'Returned from SUBPROG...'.
           DISPLAY 'WS-NUMBER: ' WS-NUMBER.
           DISPLAY 'WS-NAME: ' WS-NAME.
           DISPLAY 'WS-RESULT: ' WS-RESULT.

           STOP RUN.
```

Subprogram (SUBPROG)

This subprogram receives parameters from the main program and performs operations based on the type of parameter linkage.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SUBPROG.
DATA DIVISION.
LINKAGE SECTION.
01  LS-NUMBER        PIC 9(4).
01  LS-NAME          PIC A(20).
01  LS-INPUT-VALUE   PIC 9(4).
01  LS-RESULT        PIC 9(4).

PROCEDURE DIVISION USING LS-NUMBER LS-NAME LS-INPUT-VALUE LS-RESULT.
BEGIN-SUBPROGRAM.
    DISPLAY 'In SUBPROG...'.
    DISPLAY 'LS-NUMBER: ' LS-NUMBER.
    DISPLAY 'LS-NAME: ' LS-NAME.
    DISPLAY 'LS-INPUT-VALUE: ' LS-INPUT-VALUE.

    * Perform some operations on the passed parameters
    ADD LS-NUMBER TO LS-INPUT-VALUE GIVING LS-RESULT.

    * Modify LS-NUMBER (BY REFERENCE) to demonstrate effect on MAINPROG
    ADD 100 TO LS-NUMBER.

    DISPLAY 'Updated LS-NUMBER: ' LS-NUMBER.
    DISPLAY 'Calculated LS-RESULT: ' LS-RESULT.

    EXIT PROGRAM.
```

Explanation:

Call Statement: The main program uses the CALL statement to invoke SUBPROG.
Value Generation: Different parameter passing methods are demonstrated:
 BY REFERENCE (LS-NUMBER, LS-RESULT): Allows the subprogram to modify the caller's data directly.
 BY CONTENT (LS-NAME): Passes a copy of the data, protecting the original value in the caller.
 BY VALUE (LS-INPUT-VALUE): Passes a copy of the value, suitable for numeric parameters that do not need to be modified.
Link Parameter: Defines how data is passed between the main program and the subprogram, demonstrating the impact of each method on program behaviour.