Modularization Techniques - Subroutines

Lesson Objectives

In this lesson, participants will learn

- Subroutines

Modularization Techniques

A modularization unit is a part of a program that encapsulates a particular function.

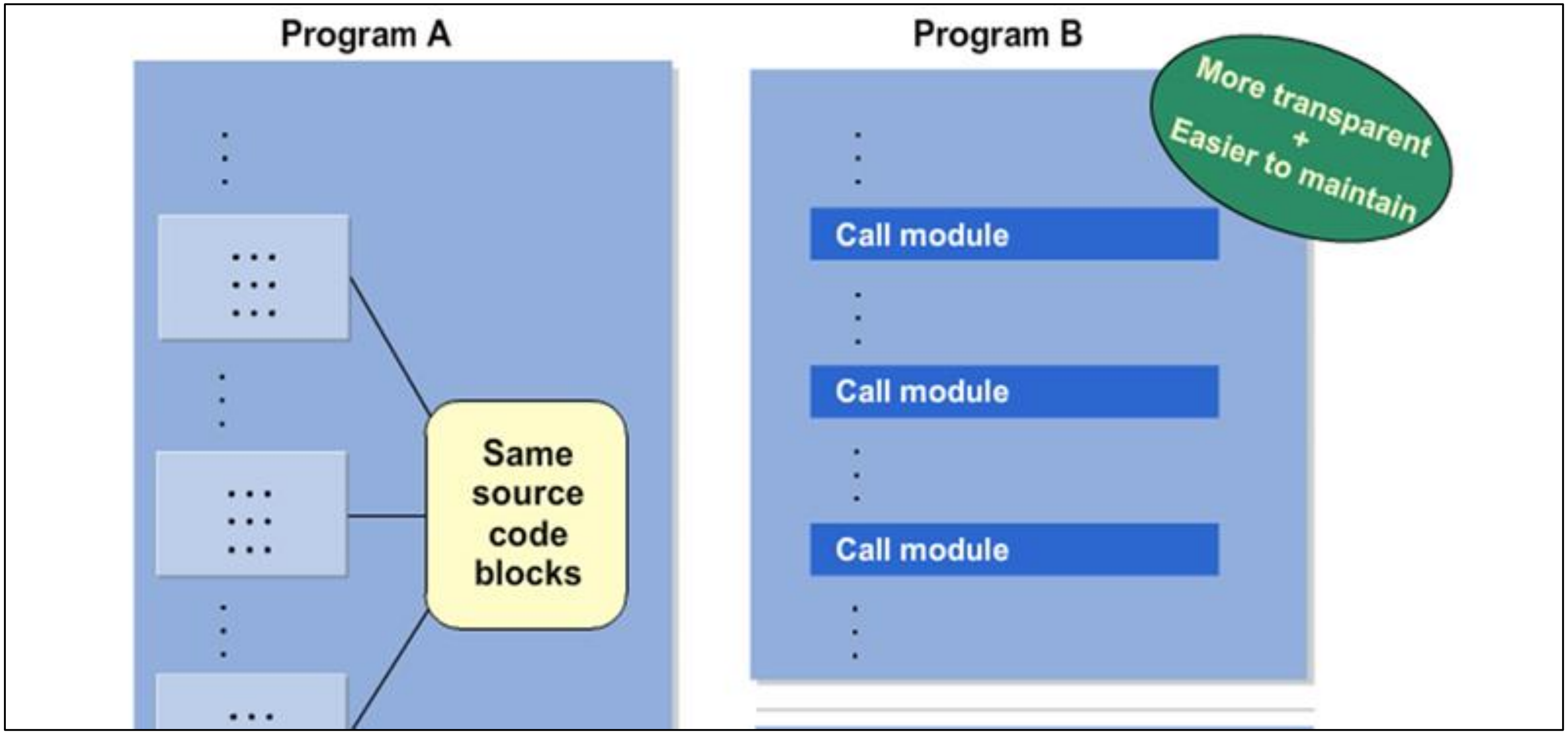
Part of the source code is stored in a module to improve the transparency of the program, as well as to use the corresponding function in the program several times without having to re-implement the entire source code on each occasion.

The improvement in transparency is a result of the program becoming more function-oriented.

It divides the overall task into sub-functions, which are the responsibility of corresponding modularization units.

Modularization makes it easier to maintain programs, because you only need to make changes to the function or corrections in the respective modularization units, and not at various points in the main program.

Modularization Techniques

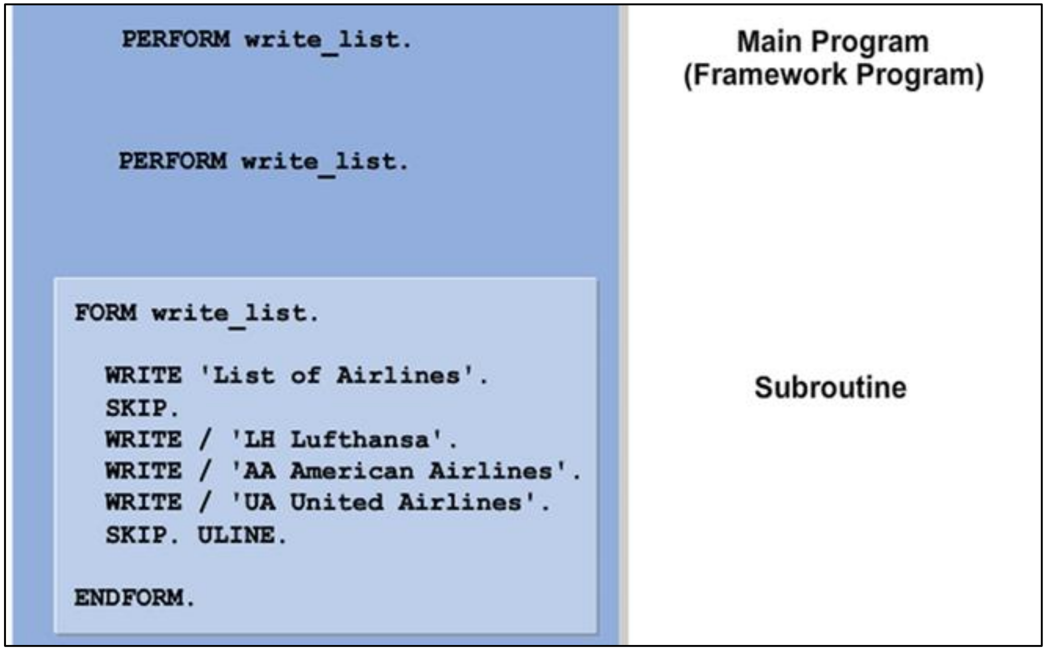


Modularization - Subroutines

Subroutines can be defined in any ABAP program.

A subroutine is a modularization unit within a program.

Subroutines are used to write functions that are used repeatedly within a program.



```
    PERFORM write_list.


    PERFORM write_list.


FORM write_list.

  WRITE 'List of Airlines'.
  SKIP.
  WRITE / 'LH Lufthansa'.
  WRITE / 'AA American Airlines'.
  WRITE / 'UA United Airlines'.
  SKIP. ULINE.

ENDFORM.
```

Modularization - Subroutines

A subroutine is a block of code introduced by FORM and concluded by ENDFORM.

Syntax

```
FORM <SUBR> [USING ... [VALUE(]<pi>[)] [TYPE <t>|LIKE <f>]... ]
[CHANGING... [VALUE(]<pi>[)] [TYPE <t>|LIKE <f>]... ].
...
ENDFORM.
```

- <SUBR> is the name of the subroutine.

Modularization - Subroutines

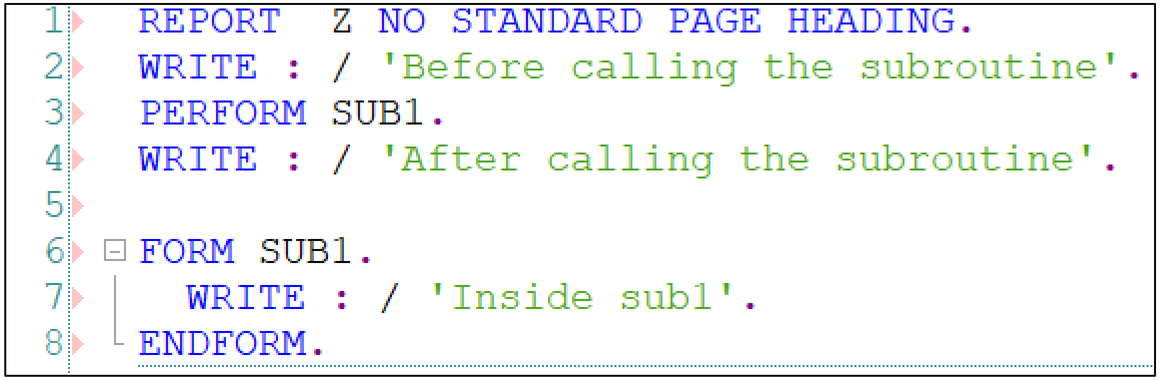
Calling Subroutines

- PERFORM………… [USING … <pi>…] [CHANGING… <pi>…].

Subroutine - Example

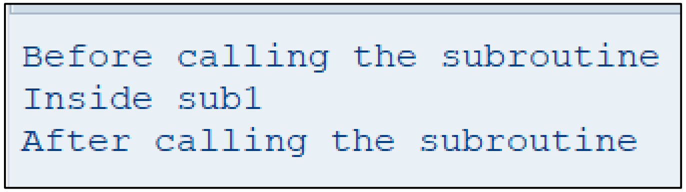
```
REPORT  Z NO STANDARD PAGE HEADING.
WRITE : / 'Before calling the subroutine'.
PERFORM SUB1.
WRITE : / 'After calling the subroutine'.

FORM SUB1.
  WRITE : / 'Inside sub1'.
ENDFORM.
```

Demo: Create Simple Subroutine



```
REPORT  Z NO STANDARD PAGE HEADING.
WRITE : / 'Before calling the subroutine'.
PERFORM SUB1.
WRITE : / 'After calling the subroutine'.

FORM SUB1.
  WRITE : / 'Inside sub1'.
ENDFORM.
```

Output after execution



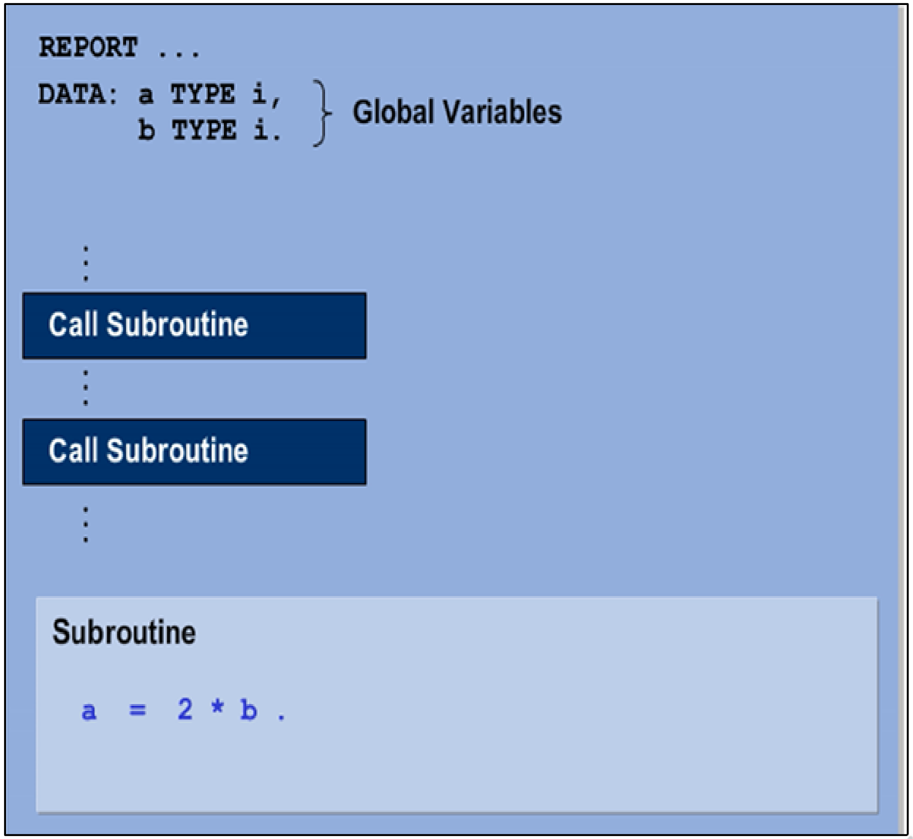
```
Before calling the subroutine
Inside sub1
After calling the subroutine
```

Parameter Definition for subroutines

Variables defined in the main program are globally visible within the program and can be changed at any point within the program.

This means that also subroutines defined within the program can change the value of these global variables.

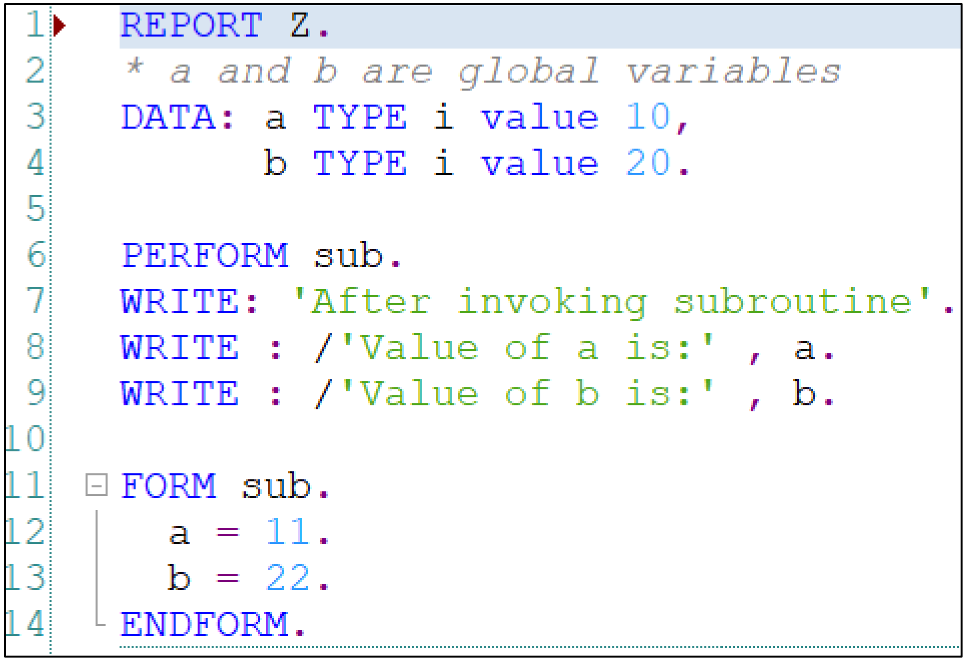
```
REPORT Z.
* a and b are global variables
DATA: a TYPE i value 10,
         b TYPE i value 20.

PERFORM sub.
WRITE: 'After invoking subroutine'.
WRITE : /'Value of a is:' , a.
WRITE : /'Value of b is:' , b.
FORM sub.
  a = 11.
  b = 22.
ENDFORM.
```



```
REPORT ...
DATA: a TYPE i,
      b TYPE i.
```

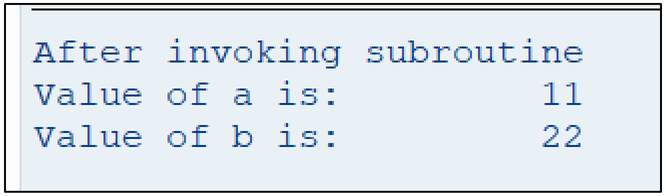
```
a = 2 * b .
```

Demo: Create Subroutine with Global Variables



```
REPORT Z.
* a and b are global variables
DATA: a TYPE i value 10,
      b TYPE i value 20.

PERFORM sub.
WRITE: 'After invoking subroutine'.
WRITE : /'Value of a is:' , a.
WRITE : /'Value of b is:' , b.

FORM sub.
  a = 11.
  b = 22.
ENDFORM.
```

Output after execution



```
After invoking subroutine
Value of a is:          11
Value of b is:          22
```

Local Variables

Variables defined by the data statement at the top of the program are global.

Variable defined inside a subroutine using the data or statics statement are local.

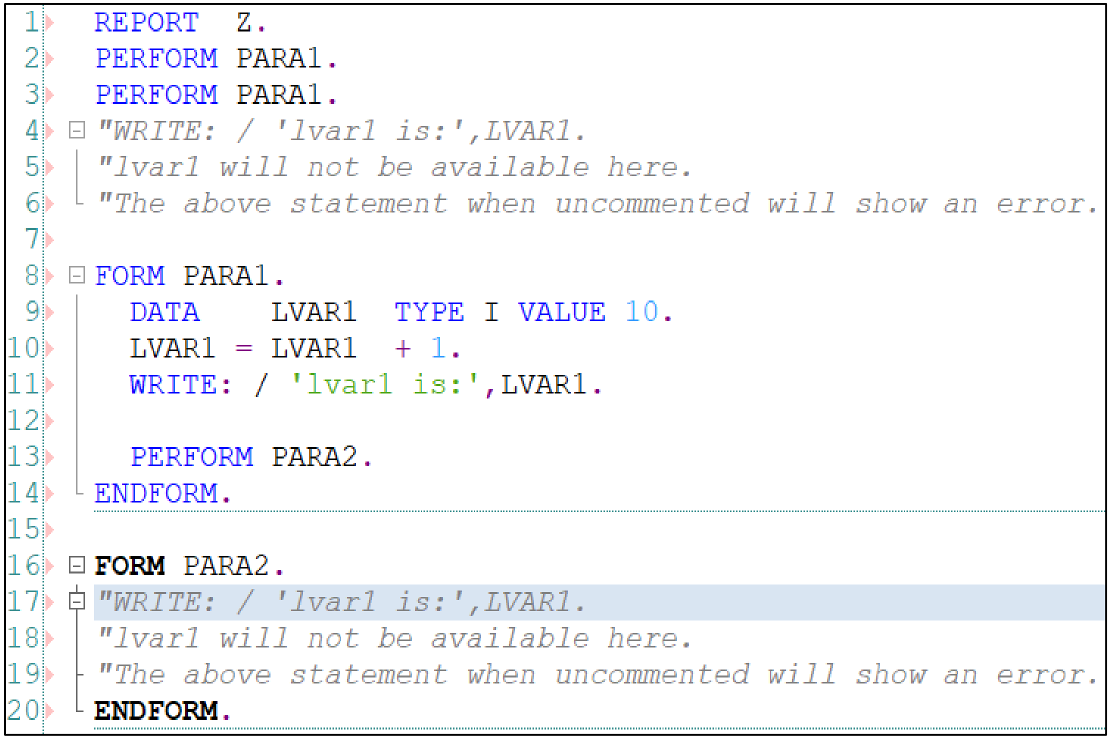
Data definitions within a subroutine are local to the subroutine.

They are not accessible from outside the subroutine.

Example – Local Variable

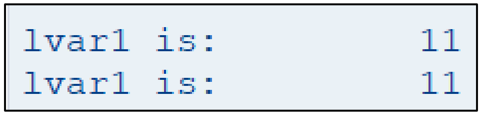
```
REPORT  Z.
PERFORM PARA1.
PERFORM PARA1.
"WRITE: / 'lvar1 is:',LVAR1.
"lvar1 will not be available here.
“The above statement when uncommented will show an error.

FORM PARA1.
  DATA    LVAR1  TYPE I VALUE 10.
  LVAR1 = LVAR1  + 1.
  WRITE: / 'lvar1 is:',LVAR1.
  PERFORM PARA2.
ENDFORM.

FORM PARA2.
"WRITE: / 'lvar1 is:',LVAR1.
"lvar1 will not be available here.
“The above statement when uncommented will show an error.
ENDFORM.
```

Demo: Create Subroutine with Local Variable



```
REPORT  Z.
PERFORM PARA1.
PERFORM PARA1.
"WRITE: / 'lvar1 is:',LVAR1.
"lvar1 will not be available here.
"The above statement when uncommented will show an error.

FORM PARA1.
  DATA    LVAR1  TYPE I VALUE 10.
  LVAR1 = LVAR1  + 1.
  WRITE: / 'lvar1 is:',LVAR1.

  PERFORM PARA2.
ENDFORM.

FORM PARA2.
"WRITE: / 'lvar1 is:',LVAR1.
"lvar1 will not be available here.
"The above statement when uncommented will show an error.
ENDFORM.
```

Output after execution



```
lvar1 is:          11
lvar1 is:          11
```

Static Variable

If you want to retain the value of a local data object after exiting the subroutine, you must use the statics statement to declare it instead of the data statement.

Use the statics statement to create local variables that are not freed when the subroutine ends.

The syntax for statics is the same as the syntax for the data statement.

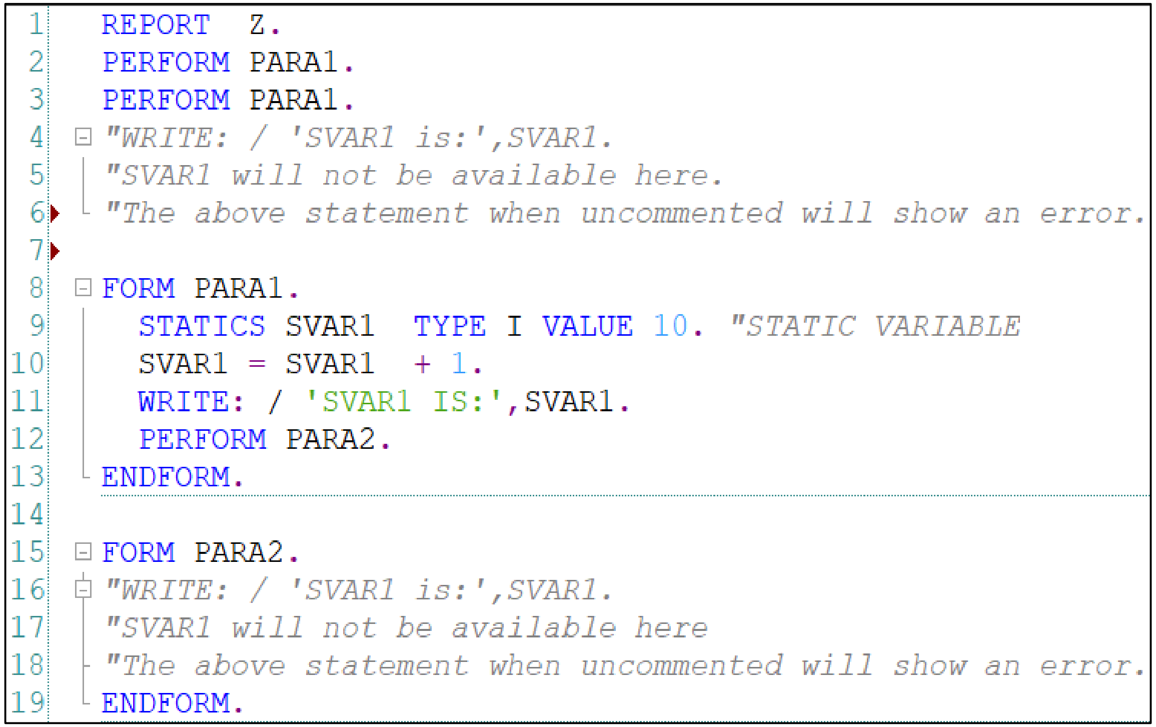
The memory for a static variable is allocated the first time the subroutine is called and retained when the subroutine ends.

The memory of a static variable belongs to the subroutine that allocated it; that variable is not visible from other subroutines

Example – Statics Variable

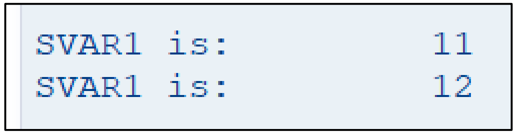
```
REPORT  Z.
PERFORM PARA1.
PERFORM PARA1.
"WRITE: / 'SVAR1 is:',SVAR1.
"SVAR1 will not be available here.
"The above statement when uncommented will show an error.

FORM PARA1.
  STATICS SVAR1  TYPE I VALUE 10. "STATIC VARIABLE
  SVAR1 = SVAR1  + 1.
  WRITE: / 'SVAR1 IS:',SVAR1.
  PERFORM PARA2.
ENDFORM.

FORM PARA2.
"WRITE: / 'SVAR1 is:',SVAR1.
"SVAR1 will not be available here
"The above statement when uncommented will show an error.
ENDFORM.
```

Demo: Create Subroutine with Static Variable



```
REPORT  Z.
PERFORM PARA1.
PERFORM PARA1.
"WRITE: / 'SVAR1 is:',SVAR1.
"SVAR1 will not be available here.
"The above statement when uncommented will show an error.

FORM PARA1.
  STATICS SVAR1  TYPE I VALUE 10. "STATIC VARIABLE
  SVAR1 = SVAR1  + 1.
  WRITE: / 'SVAR1 IS:',SVAR1.
  PERFORM PARA2.
ENDFORM.

FORM PARA2.
"WRITE: / 'SVAR1 is:',SVAR1.
"SVAR1 will not be available here
"The above statement when uncommented will show an error.
ENDFORM.
```

Output after execution



```
SVAR1 is:          11
SVAR1 is:          12
```

Passing Parameter – Definition of an Interface

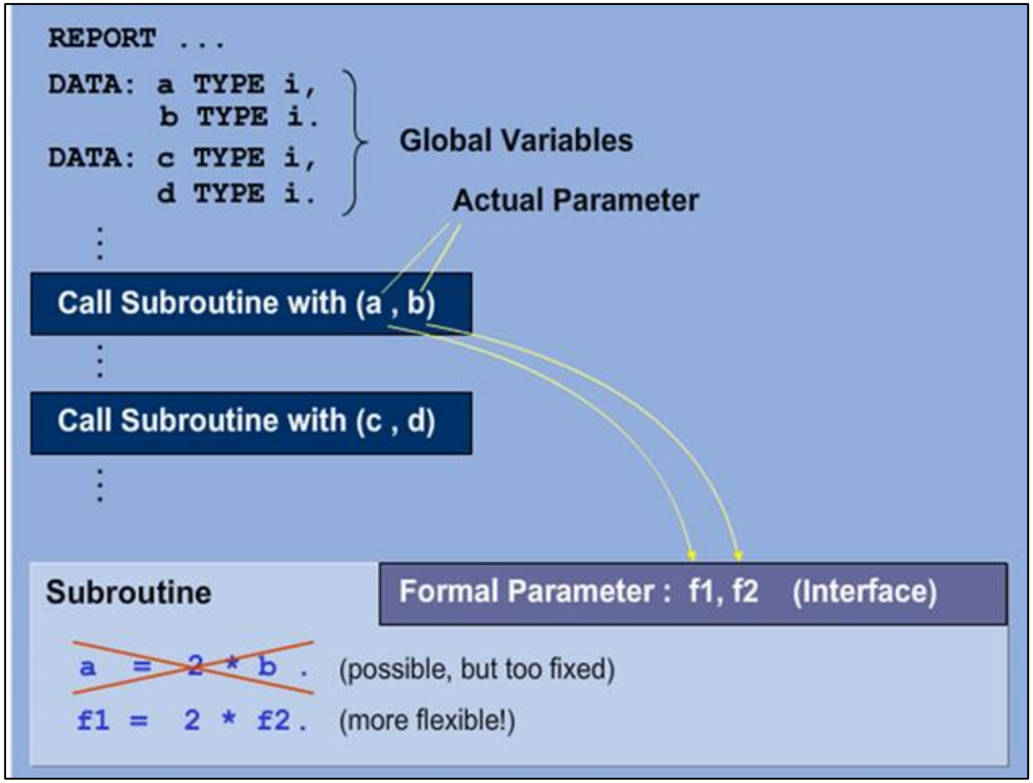
You can address all global variables defined in the main program from a subroutine.

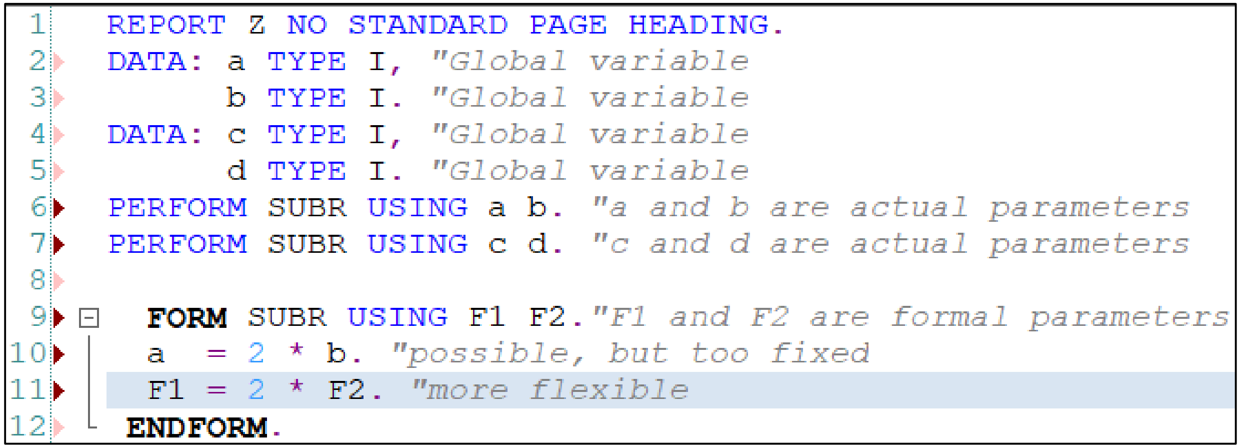
However, to call a subroutine with different data objects for each situation, you must use placeholders instead of global variables.

These placeholders are called formal parameters and together they form the subroutine interface.

You must declare the interface when you define the subroutine.

Passing Parameter – Definition of an Interface





```
REPORT Z NO STANDARD PAGE HEADING.
DATA: a TYPE I, "Global variable
      b TYPE I. "Global variable
DATA: c TYPE I, "Global variable
      d TYPE I. "Global variable
PERFORM SUBR USING a b. "a and b are actual parameters
PERFORM SUBR USING c d. "c and d are actual parameters

 FORM SUBR USING F1 F2."F1 and F2 are formal parameters
 a  = 2 * b. "possible, but too fixed
 F1 = 2 * F2. "more flexible
ENDFORM.
```

Subroutines - Parameter Interface

The **using** and **changing** additions in the FORM statement define the formal parameters of a subroutine

The global data of the report and of the system can be accessed from within the subroutine.

In addition to defining variables by using tables, data, and statics, variables can also be defined on the form statement itself.

These are known as parameters.

Parameter names that appear on the form statement are called formal parameters.

Parameter names that appear on the perform statement are called actual parameters.

- For example, in the statement perform s1 using f1 changing f2 f3, the parameters f1, f2, and f3 are called actual parameters.

When a subroutine is called, all formal parameters must be filled with the values from the actual parameters

Subroutines - Parameter Interface

Subroutines has the following formal parameters

- Parameters passed by Value
- Parameters Passed by Reference
- Passed by Value and Result

The following table shows the relationship between syntax and passing methods.

Addition	**Method**
Using v1	Pass by reference
Changing v1	Pass by reference
Using value(v1)	Pass by value
Changing value(v1)	Pass by value and result

Pass Type for subroutine

Definition of pass types for subroutines

- Pass by Value
  - The system makes a copy of the actual parameter and assigns this copy to the formal parameter.
  - Any value assignments to the corresponding formal parameter in the subroutine, therefore, refer only to the copy of actual parameter not to the original.
  - Use this pass type to make the value of a global variable available to the subroutine in the form of a variable copy, without allowing the subroutine to change the respective global variable.
  - Creating a copy of the variable value protects the original
  - However, creating copies for large internal tables can be time consuming.

Example – Pass by Value

```
REPORT Z.
DATA F1 VALUE 'A'.

WRITE: / 'Before invoking S1:',F1. "will display A
PERFORM S1 USING F1.
*PERFORM S1 CHANGING F1.
WRITE: / 'After  invoking S1:',F1. "will display A

FORM S1 USING VALUE(P1).
  P1 = 'X'.
  WRITE: / 'In Subroutine S1:',P1. "will display X
ENDFORM.
```

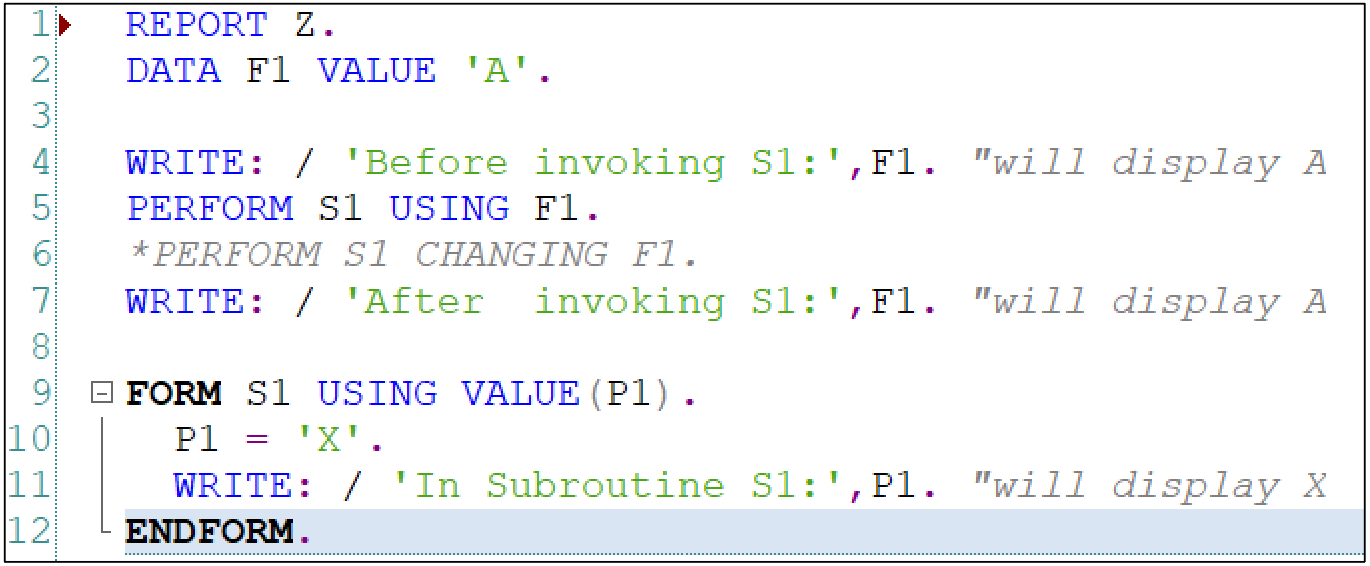
Note: In the perform statement either of the two can be used

PERFORM S1 USING F1.

PERFORM S1 CHANGING F1.

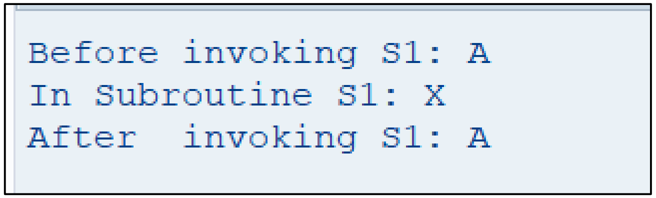
The syntax on the form statement alone determines the method by which a parameter is passed.

Demo1: Pass by Value



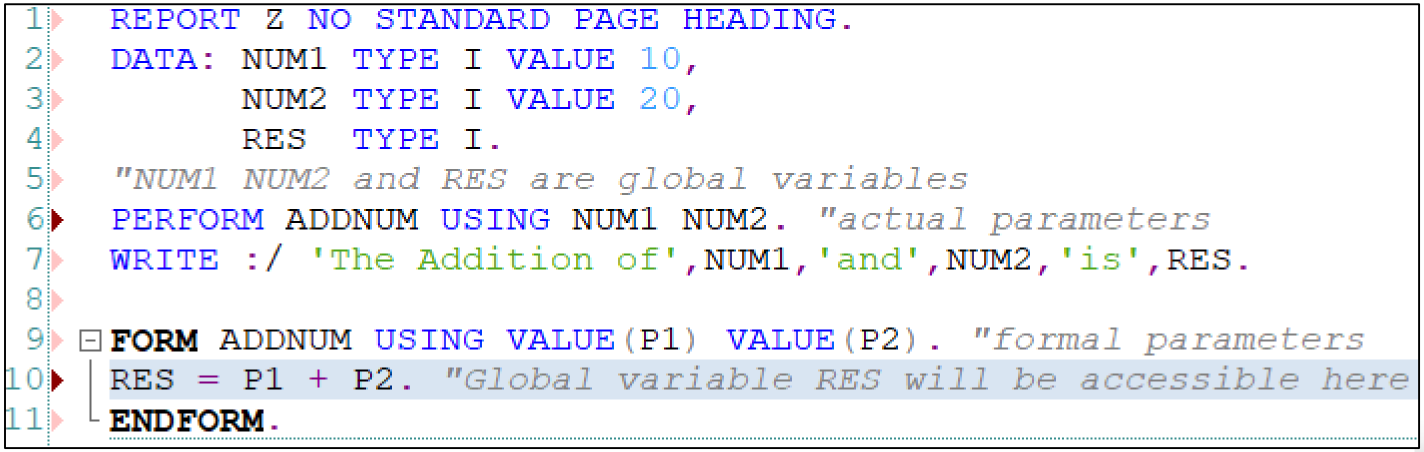
```
REPORT Z.
DATA F1 VALUE 'A'.

WRITE: / 'Before invoking S1:',F1. "will display A
PERFORM S1 USING F1.
*PERFORM S1 CHANGING F1.
WRITE: / 'After  invoking S1:',F1. "will display A

FORM S1 USING VALUE(P1).
  P1 = 'X'.
  WRITE: / 'In Subroutine S1:',P1. "will display X
ENDFORM.
```

Output after execution



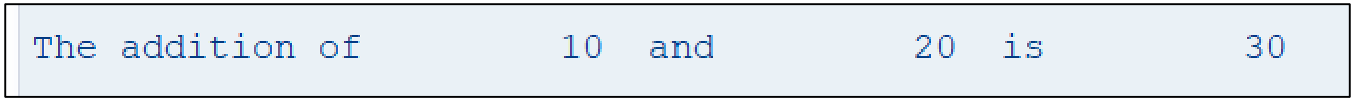
```
Before invoking S1: A
In Subroutine S1: X
After  invoking S1: A
```

Demo2: Pass by Value



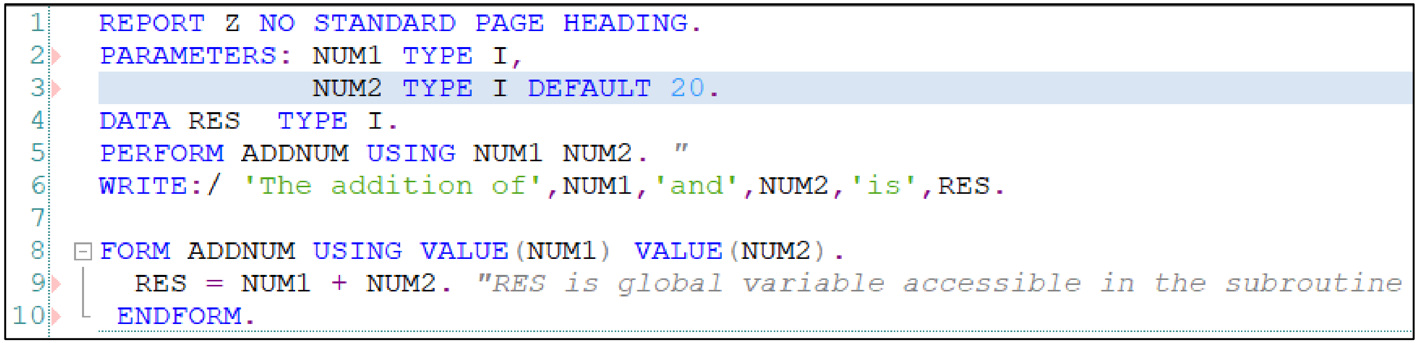
```
REPORT Z NO STANDARD PAGE HEADING.
DATA: NUM1 TYPE I VALUE 10,
      NUM2 TYPE I VALUE 20,
      RES  TYPE I.
"NUM1 NUM2 and RES are global variables
PERFORM ADDNUM USING NUM1 NUM2. "actual parameters
WRITE :/ 'The Addition of',NUM1,'and',NUM2,'is',RES.

FORM ADDNUM USING VALUE(P1) VALUE(P2). "formal parameters
RES = P1 + P2. "Global variable RES will be accessible here
ENDFORM.
```

Output after execution



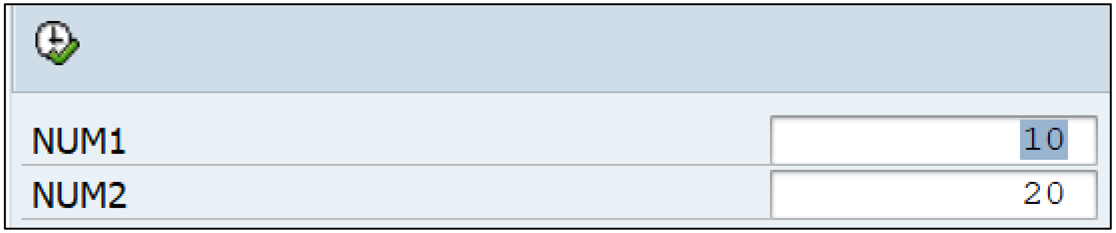
```
The addition of          10  and          20  is          30
```

Demo3: Pass by Value



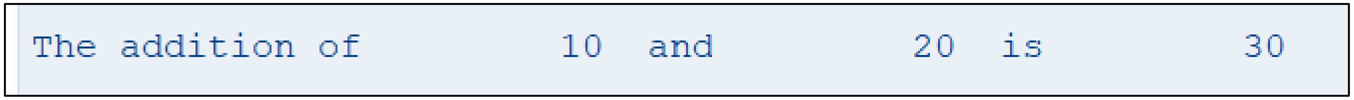
```
REPORT Z NO STANDARD PAGE HEADING.
PARAMETERS: NUM1 TYPE I,
            NUM2 TYPE I DEFAULT 20.
DATA RES  TYPE I.
PERFORM ADDNUM USING NUM1 NUM2. "
WRITE:/ 'The addition of',NUM1,'and',NUM2,'is',RES.

FORM ADDNUM USING VALUE(NUM1) VALUE(NUM2).
  RES = NUM1 + NUM2. "RES is global variable accessible in the subroutine
 ENDFORM.
```

Selection screen



NUM1	10
NUM2	20

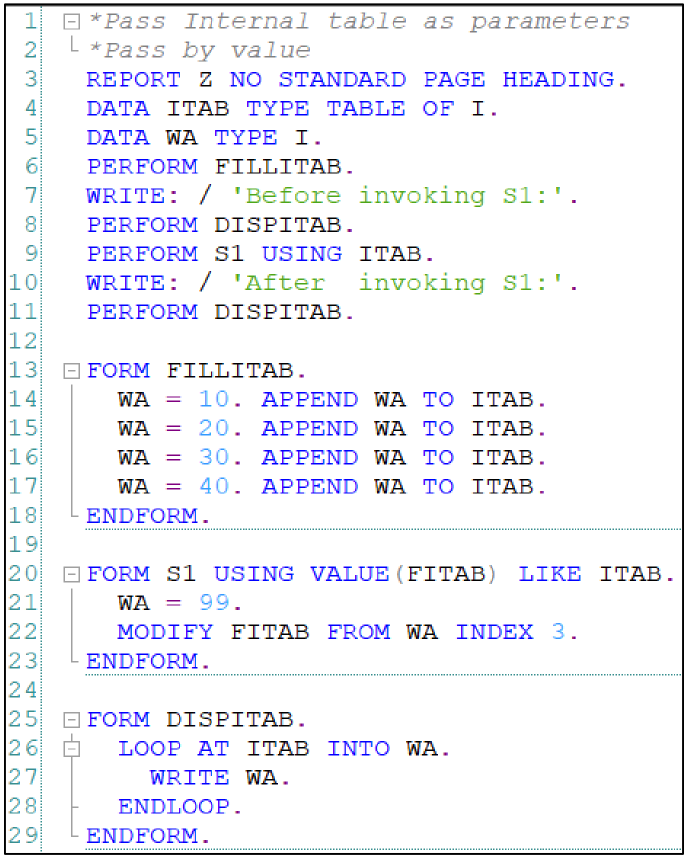
Output after execution



```
The addition of          10  and          20  is          30
```

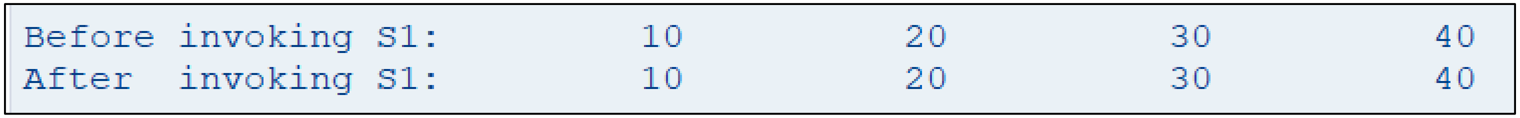
Note: Num1 and Num2 can have different values since they are parameters

Demo1: Pass by Value – Internal Table



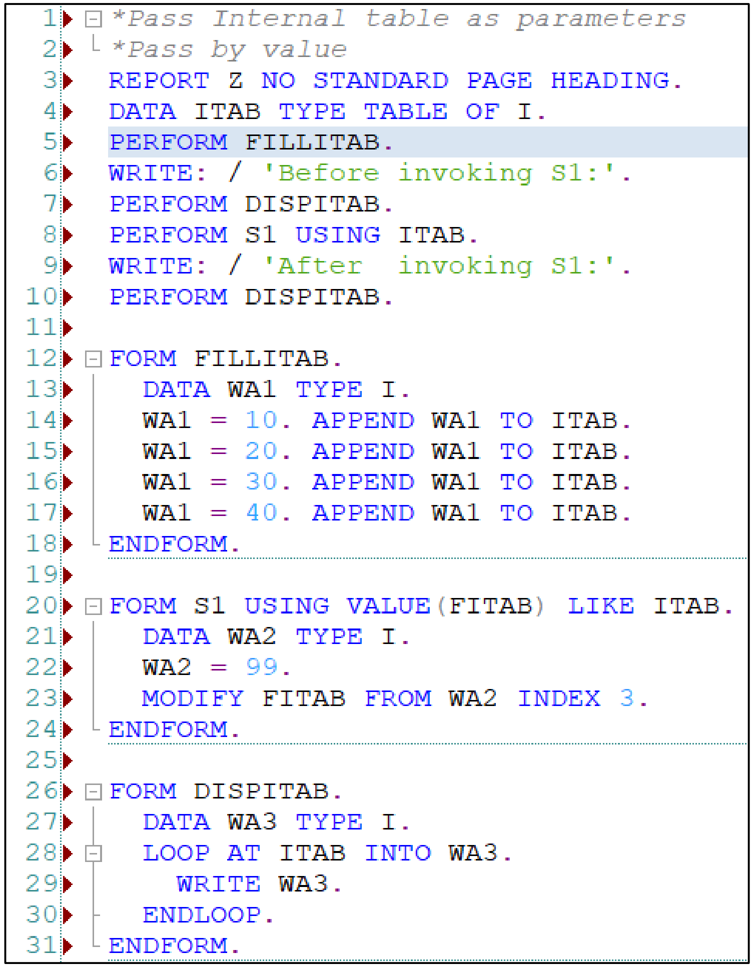
```
*Pass Internal table as parameters
*Pass by value
REPORT Z NO STANDARD PAGE HEADING.
DATA ITAB TYPE TABLE OF I.
DATA WA TYPE I.
PERFORM FILLITAB.
WRITE: / 'Before invoking S1:'.
PERFORM DISPITAB.
PERFORM S1 USING ITAB.
WRITE: / 'After  invoking S1:'.
PERFORM DISPITAB.

FORM FILLITAB.
  WA = 10. APPEND WA TO ITAB.
  WA = 20. APPEND WA TO ITAB.
  WA = 30. APPEND WA TO ITAB.
  WA = 40. APPEND WA TO ITAB.
ENDFORM.

FORM S1 USING VALUE(FITAB) LIKE ITAB.
  WA = 99.
  MODIFY FITAB FROM WA INDEX 3.
ENDFORM.

FORM DISPITAB.
  LOOP AT ITAB INTO WA.
    WRITE WA.
  ENDLOOP.
ENDFORM.
```

Output after execution



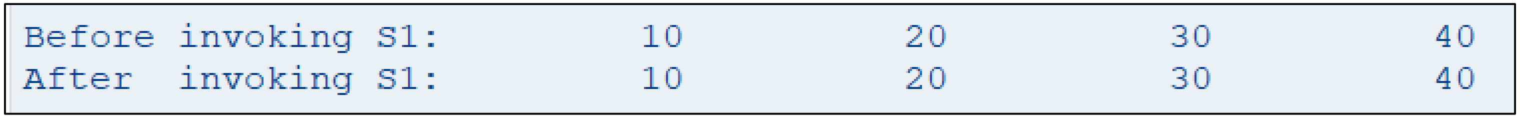
```
Before invoking S1:          10          20          30          40
After  invoking S1:          10          20          30          40
```

Demo2: Pass by Value – Internal Table



```
*Pass Internal table as parameters
*Pass by value
REPORT Z NO STANDARD PAGE HEADING.
DATA ITAB TYPE TABLE OF I.
PERFORM FILLITAB.
WRITE: / 'Before invoking S1:'.
PERFORM DISPITAB.
PERFORM S1 USING ITAB.
WRITE: / 'After  invoking S1:'.
PERFORM DISPITAB.

FORM FILLITAB.
  DATA WA1 TYPE I.
  WA1 = 10. APPEND WA1 TO ITAB.
  WA1 = 20. APPEND WA1 TO ITAB.
  WA1 = 30. APPEND WA1 TO ITAB.
  WA1 = 40. APPEND WA1 TO ITAB.
ENDFORM.

FORM S1 USING VALUE(FITAB) LIKE ITAB.
  DATA WA2 TYPE I.
  WA2 = 99.
  MODIFY FITAB FROM WA2 INDEX 3.
ENDFORM.

FORM DISPITAB.
  DATA WA3 TYPE I.
  LOOP AT ITAB INTO WA3.
    WRITE WA3.
  ENDLOOP.
ENDFORM.
```

Output after execution



```
Before invoking S1:          10          20          30          40
After  invoking S1:          10          20          30          40
```

Subroutines - Parameter Interface

The syntax on form and perform can differ.

However, for the sake of program clarity they should be same.

When calling a subroutine (perform), you can also distinguish between using and changing parameter.

However, the distinction is used for documentation purposes only.

What counts is the declaration when you define the subroutine (FORM).

The following things must be kept in mind:

- Perform and form statement must contain the same number of parameters
- The syntax on the form statement alone determines the method by which a parameter is passed
- The addition using can only occur once in a statement.
- The same rule applies to changing

Pass Type for subroutine

Definition of pass types for subroutines

- Pass by reference
  - The system assigns the actual parameter directly to the formal parameter.
  - The value assignments to the formal parameter are carried out directly on the actual parameter.
  - This pass type can be used to run subroutine processing directly on the specified actual parameter.
  - It is a useful way of avoiding the time-consuming creation of copies for large internal tables.

Example – Pass by reference

```
REPORT Z.
DATA F1 VALUE 'A'.

WRITE: / 'Before invoking S1:',F1. "will display A
PERFORM S1 CHANGING F1.
WRITE: / 'After  invoking S1:',F1. "will display X

FORM S1 CHANGING P1.
  P1 = 'X'.
  WRITE: / 'In Subroutine S1:',P1. "will display X
ENDFORM.
```

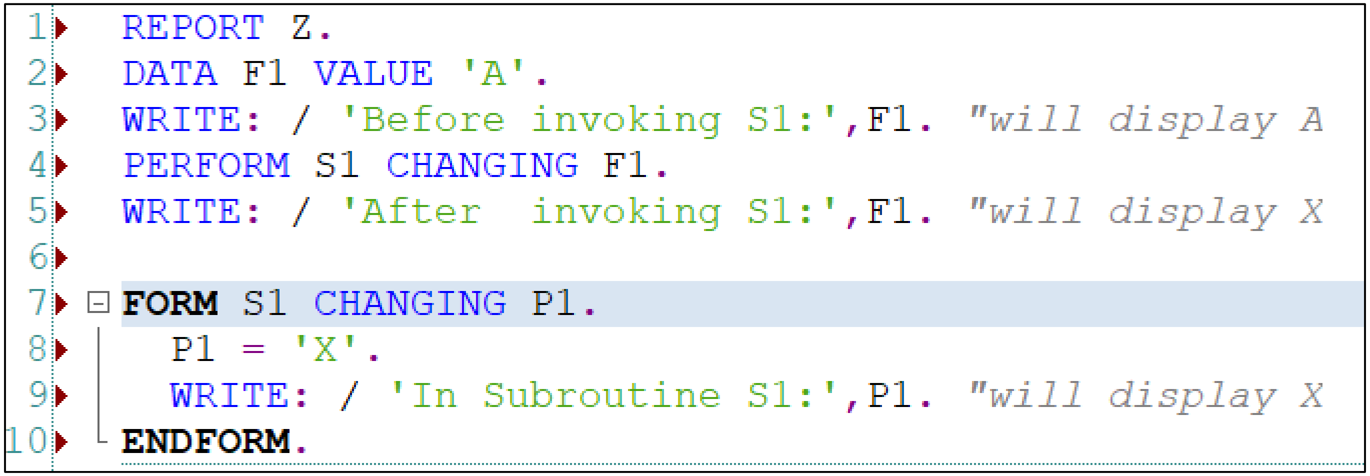
Note: In the perform statement either f the two can be used

```
PERFORM S1 USING F1.
```

```
PERFORM S1 CHANGING F1.
```

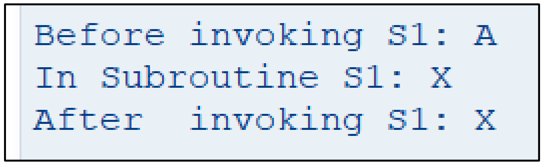
The syntax on the form statement alone determines the method by which a parameter is passed.

Demo1: Pass by Reference



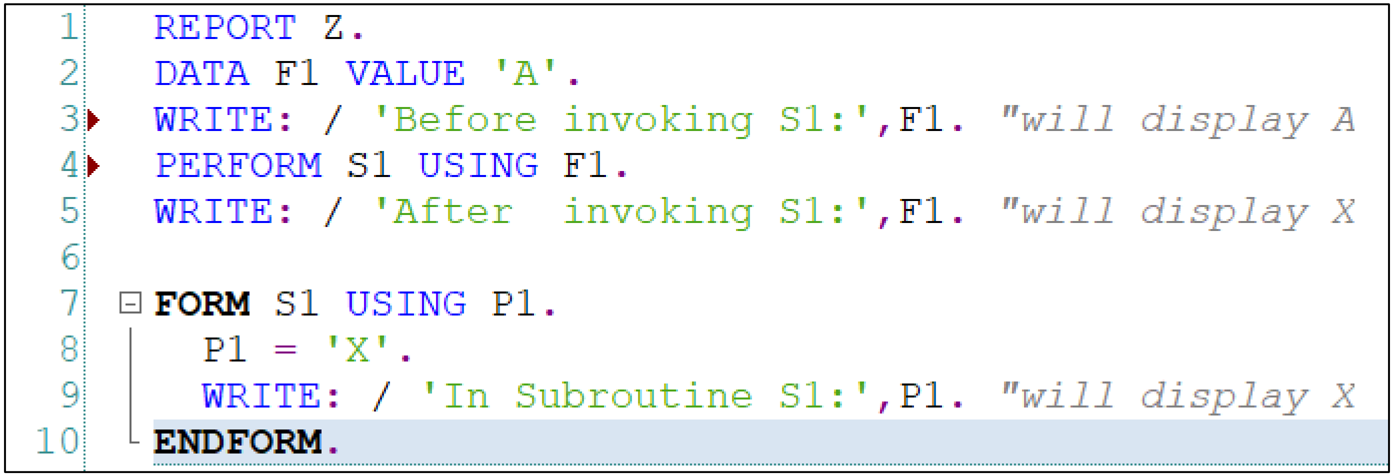
```
REPORT Z.
DATA F1 VALUE 'A'.
WRITE: / 'Before invoking S1:',F1. "will display A
PERFORM S1 CHANGING F1.
WRITE: / 'After  invoking S1:',F1. "will display X

FORM S1 CHANGING P1.
  P1 = 'X'.
  WRITE: / 'In Subroutine S1:',P1. "will display X
ENDFORM.
```

Output after execution



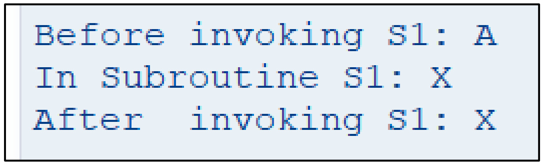
```
Before invoking S1: A
In Subroutine S1: X
After  invoking S1: X
```

Demo2: Pass by Reference



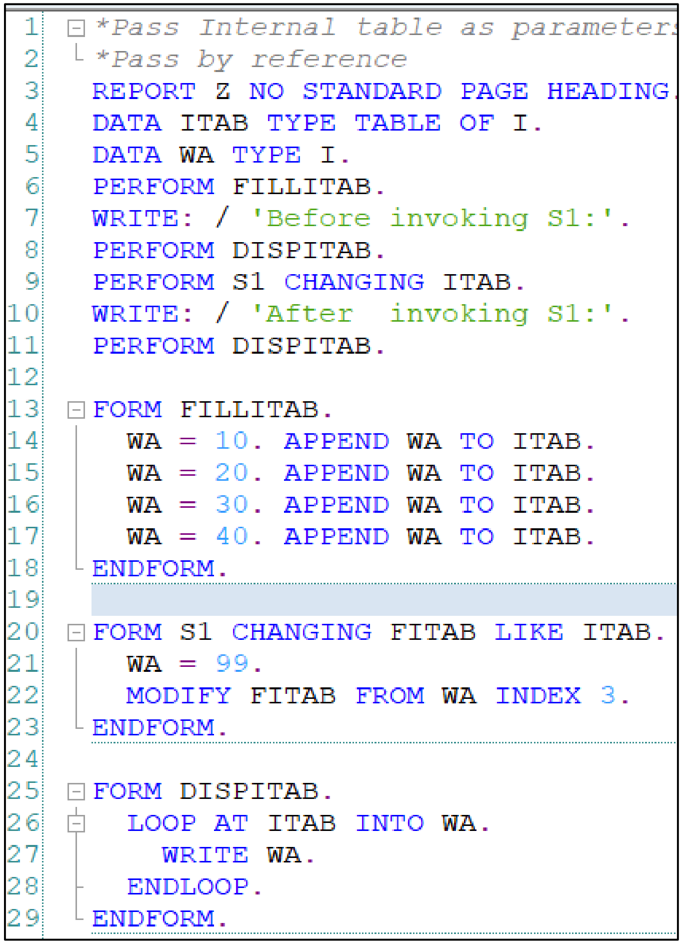
```
REPORT Z.
DATA F1 VALUE 'A'.
WRITE: / 'Before invoking S1:',F1. "will display A
PERFORM S1 USING F1.
WRITE: / 'After  invoking S1:',F1. "will display X

FORM S1 USING P1.
  P1 = 'X'.
  WRITE: / 'In Subroutine S1:',P1. "will display X
ENDFORM.
```

Output after execution



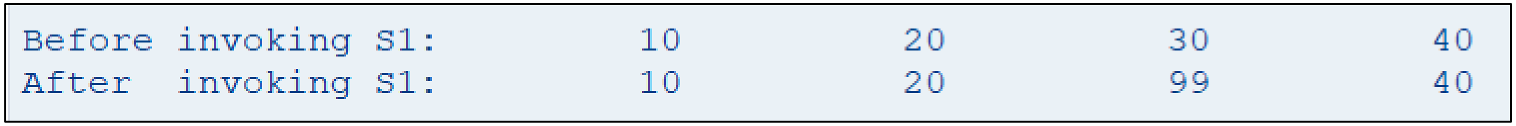
```
Before invoking S1: A
In Subroutine S1: X
After  invoking S1: X
```

Demo1: Pass by Reference – Internal Table



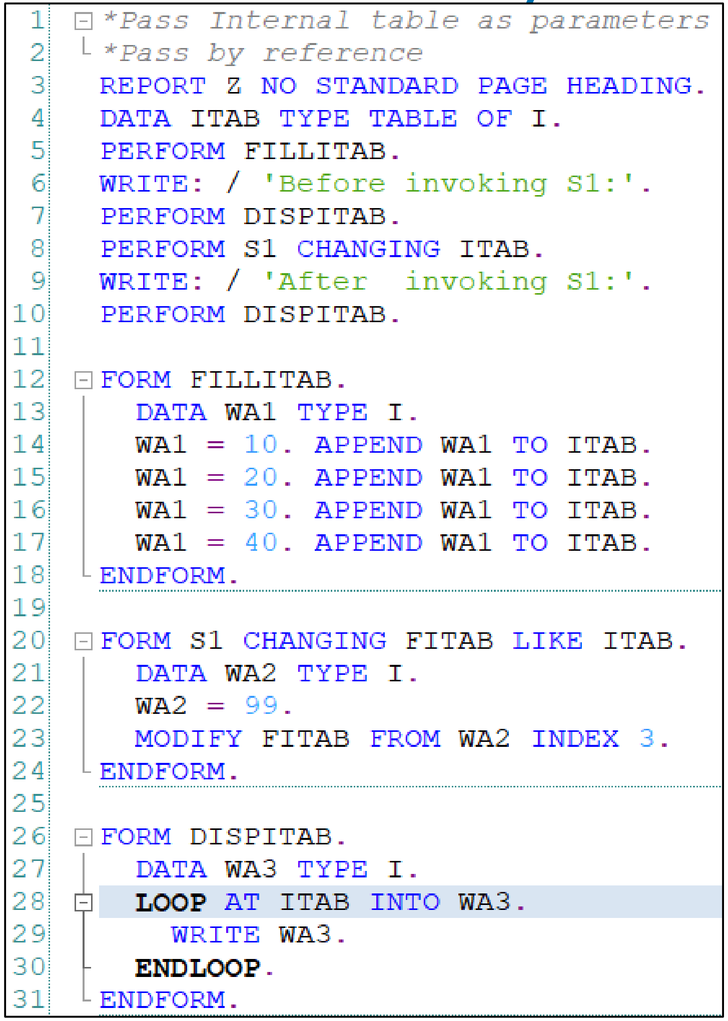
```
*Pass Internal table as parameter
*Pass by reference
REPORT Z NO STANDARD PAGE HEADING.
DATA ITAB TYPE TABLE OF I.
DATA WA TYPE I.
PERFORM FILLITAB.
WRITE: / 'Before invoking S1:'.
PERFORM DISPITAB.
PERFORM S1 CHANGING ITAB.
WRITE: / 'After  invoking S1:'.
PERFORM DISPITAB.

FORM FILLITAB.
  WA = 10. APPEND WA TO ITAB.
  WA = 20. APPEND WA TO ITAB.
  WA = 30. APPEND WA TO ITAB.
  WA = 40. APPEND WA TO ITAB.
ENDFORM.

FORM S1 CHANGING FITAB LIKE ITAB.
  WA = 99.
  MODIFY FITAB FROM WA INDEX 3.
ENDFORM.

FORM DISPITAB.
  LOOP AT ITAB INTO WA.
    WRITE WA.
  ENDLOOP.
ENDFORM.
```

Output after execution



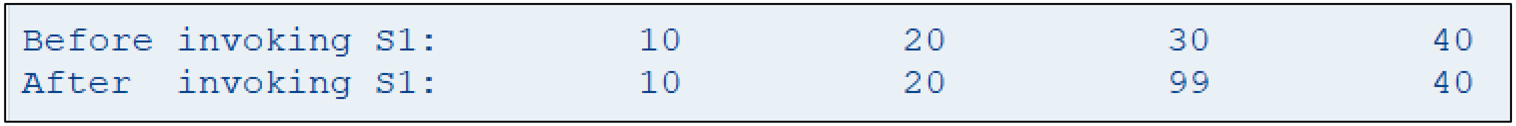
```
Before invoking S1:         10          20          30          40
After  invoking S1:         10          20          99          40
```

Demo2: Pass by Reference – Internal Table



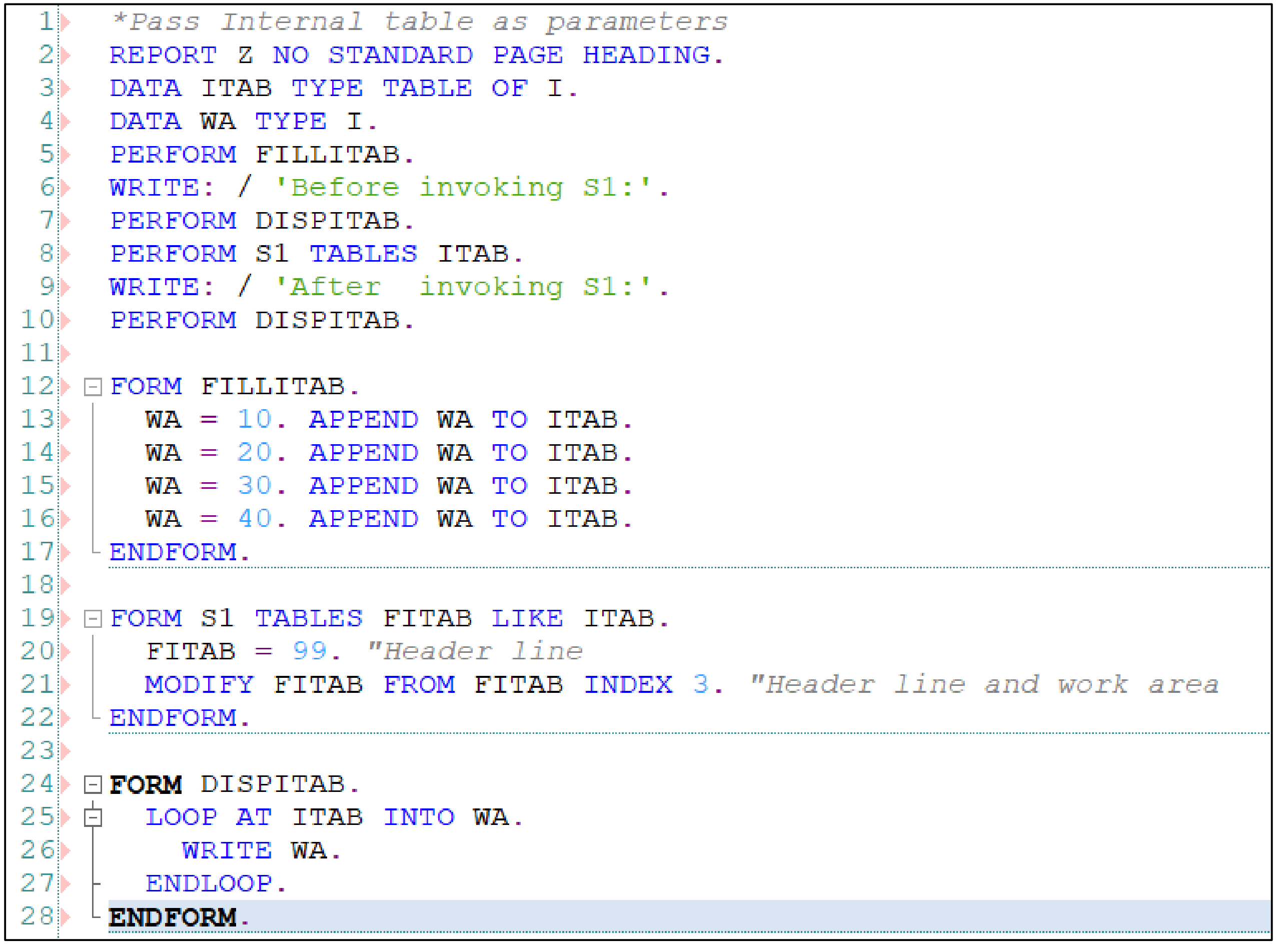
```
*Pass Internal table as parameters
*Pass by reference
REPORT Z NO STANDARD PAGE HEADING.
DATA ITAB TYPE TABLE OF I.
PERFORM FILLITAB.
WRITE: / 'Before invoking S1:'.
PERFORM DISPITAB.
PERFORM S1 CHANGING ITAB.
WRITE: / 'After  invoking S1:'.
PERFORM DISPITAB.

FORM FILLITAB.
  DATA WA1 TYPE I.
  WA1 = 10. APPEND WA1 TO ITAB.
  WA1 = 20. APPEND WA1 TO ITAB.
  WA1 = 30. APPEND WA1 TO ITAB.
  WA1 = 40. APPEND WA1 TO ITAB.
ENDFORM.

FORM S1 CHANGING FITAB LIKE ITAB.
  DATA WA2 TYPE I.
  WA2 = 99.
  MODIFY FITAB FROM WA2 INDEX 3.
ENDFORM.

FORM DISPITAB.
  DATA WA3 TYPE I.
  LOOP AT ITAB INTO WA3.
    WRITE WA3.
  ENDLOOP.
ENDFORM.
```

Output after execution



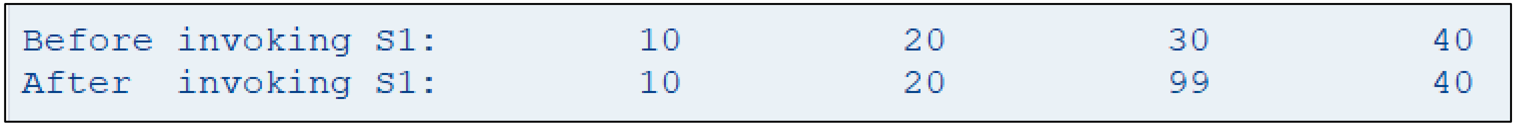
```
Before invoking S1:          10           20           30           40
After  invoking S1:          10           20           99           40
```

Demo3: Pass by Reference – Internal Table



```
*Pass Internal table as parameters
REPORT Z NO STANDARD PAGE HEADING.
DATA ITAB TYPE TABLE OF I.
DATA WA TYPE I.
PERFORM FILLITAB.
WRITE: / 'Before invoking S1:'.
PERFORM DISPITAB.
PERFORM S1 TABLES ITAB.
WRITE: / 'After  invoking S1:'.
PERFORM DISPITAB.

FORM FILLITAB.
  WA = 10. APPEND WA TO ITAB.
  WA = 20. APPEND WA TO ITAB.
  WA = 30. APPEND WA TO ITAB.
  WA = 40. APPEND WA TO ITAB.
ENDFORM.

FORM S1 TABLES FITAB LIKE ITAB.
  FITAB = 99. "Header line
  MODIFY FITAB FROM FITAB INDEX 3. "Header line and work area
ENDFORM.

FORM DISPITAB.
  LOOP AT ITAB INTO WA.
    WRITE WA.
  ENDLOOP.
ENDFORM.
```

Output after execution



```
Before invoking S1:          10          20          30          40
After  invoking S1:          10          20          99          40
```

Using .. Changing

The additions using f1 and changing f1 both pass f1 by reference-they are identical in function.

The reason they both exist is that-used properly-they can document whether the subroutine will change a parameter or not.

Code changing with parameters, the subroutine changes.

Code using with the parameters that are not changed by the subroutine

Subroutines – Additional Reading

Lesson Objectives

In this lesson, participants will learn

- Subroutines

Leaving a Subroutine

You can exit of a subroutine at any time using the following statements:

- exit
- check
- Stop

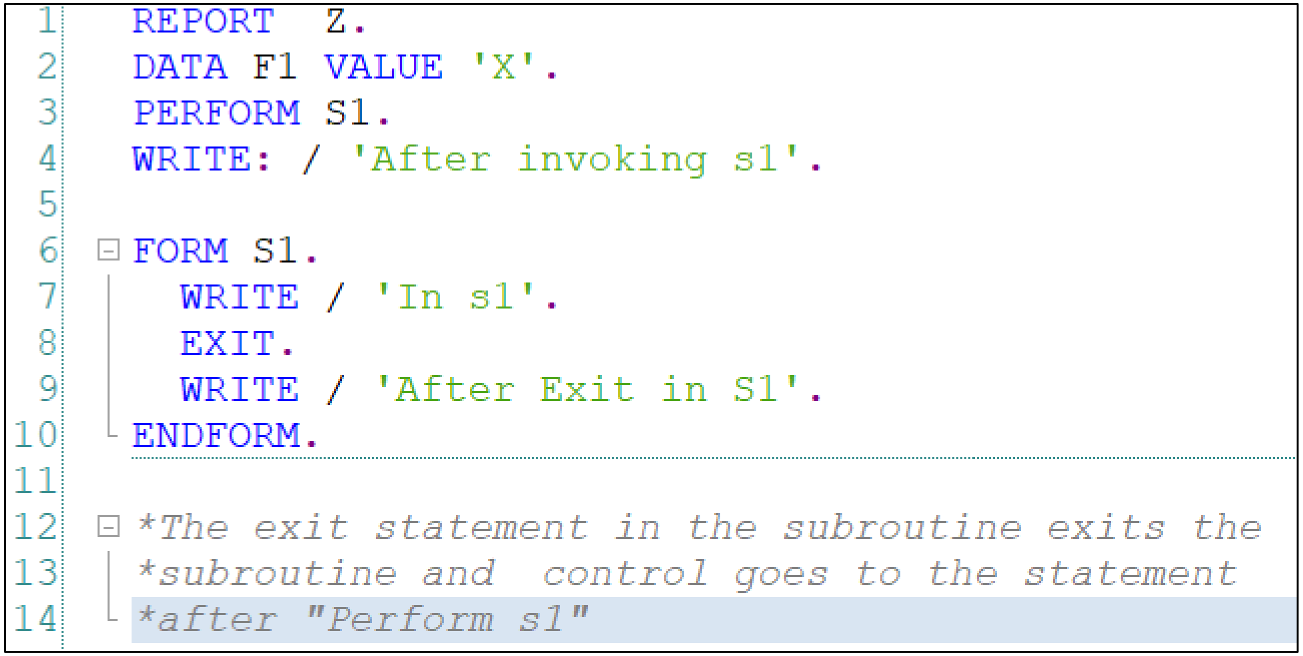
In subroutines

- check and exit immediately leave the subroutine and processing continues with the next executable statement following the perform
- stop immediately leaves the subroutine and goes directly to the end-of-selection event

Leaving subroutine using Exit - Example

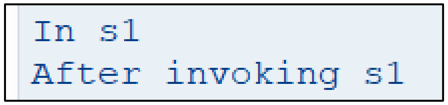
```
REPORT  Z.
DATA F1 VALUE 'X'.
PERFORM S1.
WRITE: / 'After invoking s1'.

FORM S1.
  WRITE / 'In s1'.
  EXIT.
  WRITE / 'After Exit in S1'.
ENDFORM.

*The exit statement in the subroutine exits the
*subroutine and  control goes to the statement
*after "Perform s1"
```

Demo: Leaving subroutine using Exit



```
REPORT  Z.
DATA F1 VALUE 'X'.
PERFORM S1.
WRITE: / 'After invoking s1'.

FORM S1.
  WRITE / 'In s1'.
  EXIT.
  WRITE / 'After Exit in S1'.
ENDFORM.

*The exit statement in the subroutine exits the
*subroutine and  control goes to the statement
*after "Perform s1"
```

Output after execution



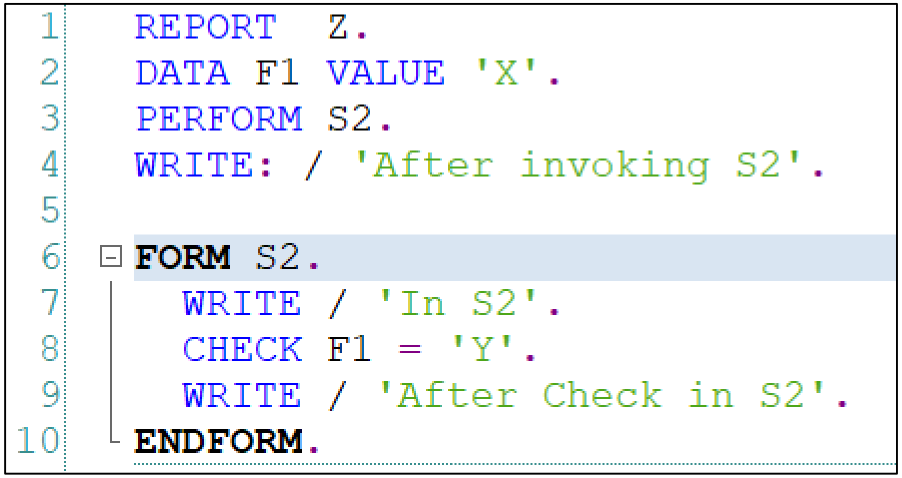
```
In s1
After invoking s1
```

Leaving subroutine using Check - Example

```
REPORT  Z.
DATA F1 VALUE 'X'.
PERFORM S2.
WRITE: / 'After invoking S2'.

FORM S2.
  WRITE / 'In S2'.
  CHECK F1  = 'Y'.
  WRITE / 'After Check in S2'.
ENDFORM.

*In form s2, when the statement check F1 = 'Y', is encountered
*control comes out of the subroutine, since the check statement leaves the
subroutine.
*Use check to terminate a subroutine conditionally.
*The subroutine is terminated if the logical expression in the CHECK statem
ent IS untrue,
*and the calling program resumes processing after the PERFORM statement.
```

Demo: Leaving subroutine using Check

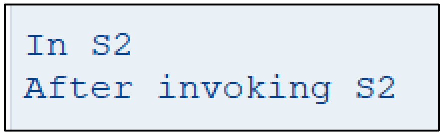
Logical expression in the CHECK statement is untrue



```
REPORT  Z.
DATA F1 VALUE 'X'.
PERFORM S2.
WRITE: / 'After invoking S2'.

FORM S2.
  WRITE / 'In S2'.
  CHECK F1 = 'Y'.
  WRITE / 'After Check in S2'.
ENDFORM.
```

- *If the logical expression in the CHECK statement IS untrue,*
- *the subroutine is terminated*

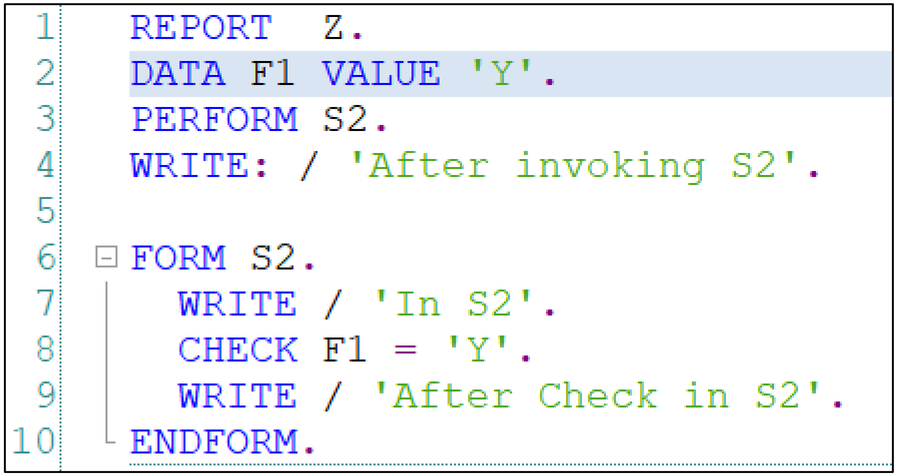
Output after execution



```
In S2
After invoking S2
```

Demo: Leaving subroutine using Check

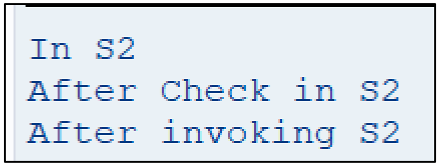
Logical expression in the CHECK statement is true



```
REPORT  Z.
DATA F1 VALUE 'Y'.
PERFORM S2.
WRITE: / 'After invoking S2'.

FORM S2.
  WRITE / 'In S2'.
  CHECK F1 = 'Y'.
  WRITE / 'After Check in S2'.
ENDFORM.
```

- *If the logical expression in the CHECK statement IS true,*
- *the subroutine is NOT terminated*

Output after execution



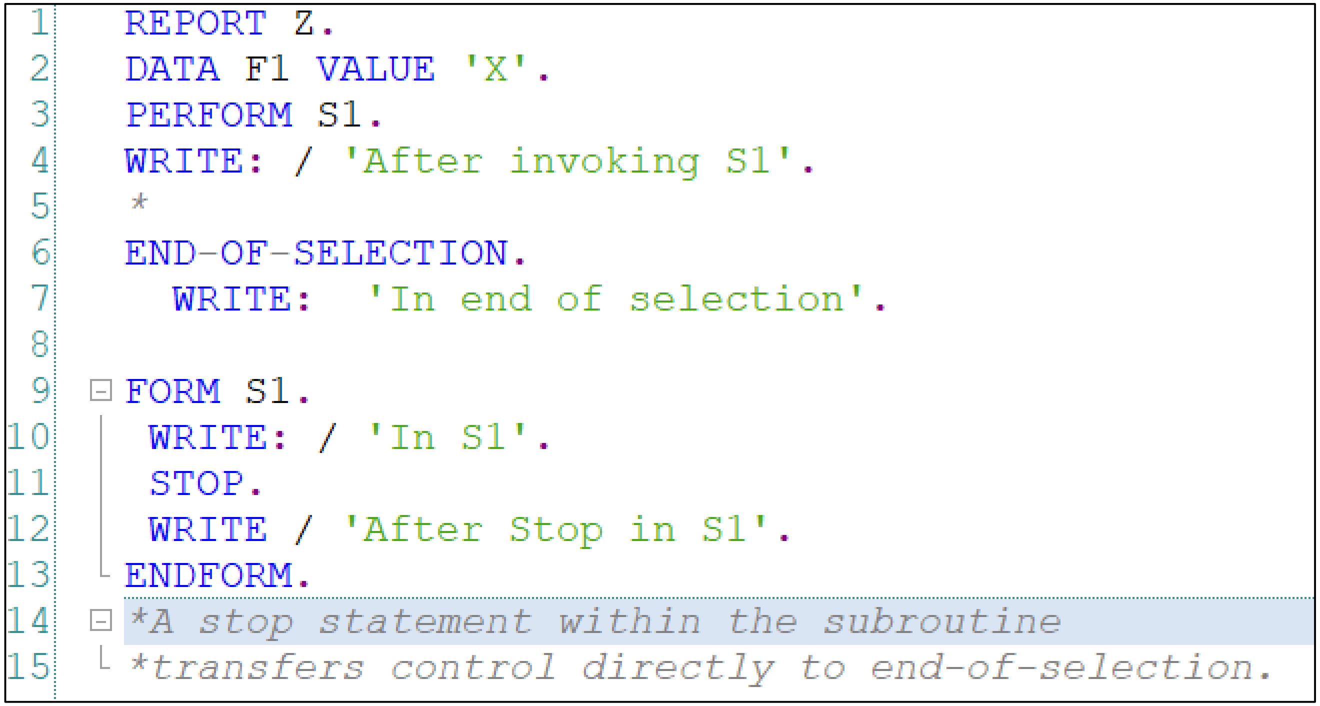
```
In S2
After Check in S2
After invoking S2
```

Leaving subroutine using Stop - Example

```
REPORT Z.
DATA F1 VALUE 'X'.
PERFORM S1.
WRITE: / 'After invoking S1'.
*
END-OF-SELECTION.
  WRITE:  'In end of selection'.

FORM S1.
 WRITE: / 'In S1'.
 STOP.
 WRITE / 'After Stop in S1'.
ENDFORM.
```

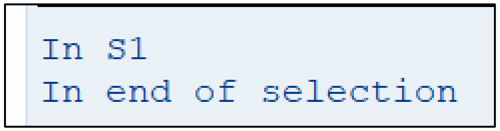
*A stop statement within the subroutine transfers control directly to end-of-selection.

Demo: Leaving subroutine using stop



```
REPORT Z.
DATA F1 VALUE 'X'.
PERFORM S1.
WRITE: / 'After invoking S1'.
*
END-OF-SELECTION.
  WRITE:  'In end of selection'.

FORM S1.
 WRITE: / 'In S1'.
 STOP.
 WRITE / 'After Stop in S1'.
ENDFORM.
*A stop statement within the subroutine
*transfers control directly to end-of-selection.
```

Output after execution



```
In S1
In end of selection
```

Passing Parameters by Value and Result

Pass by value and result is very similar to pass by value.

Like pass by value, a new memory area is allocated and it holds an independent copy of the variable.

It is freed when the subroutine ends, and that is also when the difference occurs.

When the endform statement executes, it copies the value of the local memory area back into the original memory area.

Changes to the parameter within the subroutine are reflected in the original, but not until the subroutine returns

This may seem like a small difference, but the difference becomes greater.

You can change whether the copy takes place or not.

The copy always takes place unless you leave the subroutine by using one of the two statements:

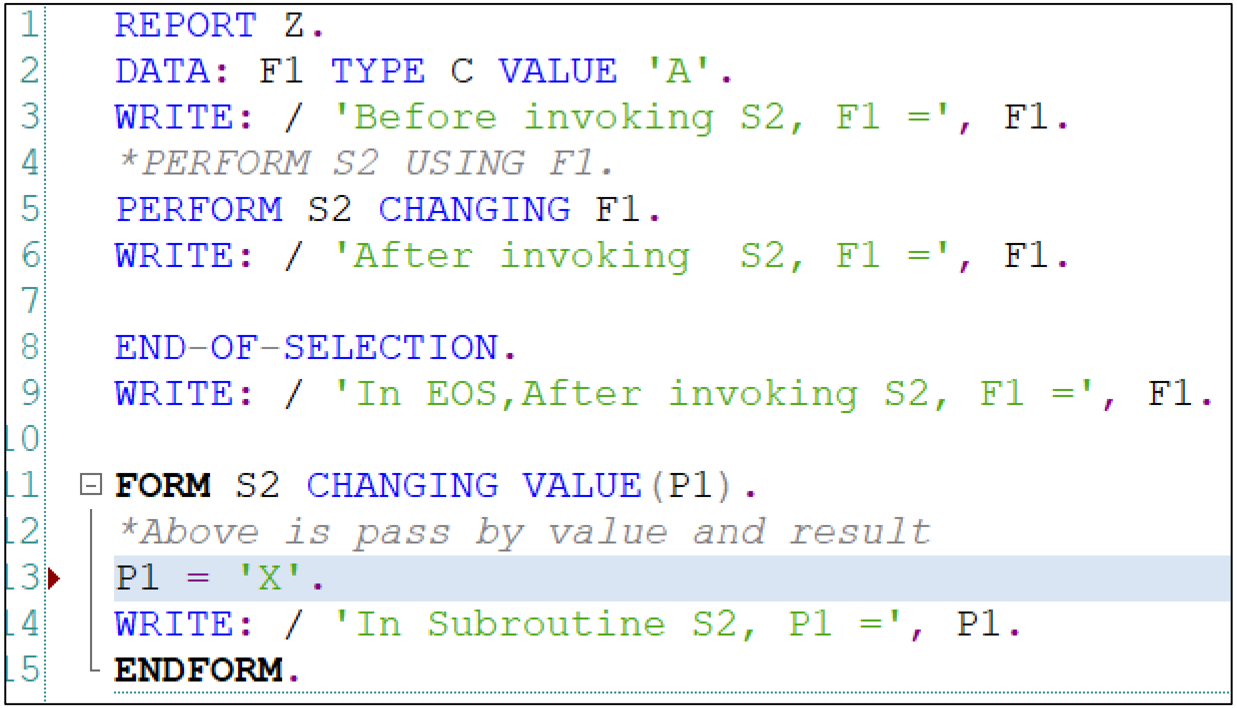
- Stop
- Message ennn(error message)

Passing Parameters by Value and Result - Example

```
REPORT Z.
DATA: F1 TYPE C VALUE 'A'.
WRITE: / 'Before invoking S2, F1 =', F1.
*PERFORM S2 USING F1.
PERFORM S2 CHANGING F1.
WRITE: / 'After invoking  S2, F1 =', F1.

END-OF-SELECTION.
WRITE: / 'In EOS, After invoking S2, F1 =', F1.

FORM S2 CHANGING VALUE(P1).
*Above is pass by value and result
P1 = 'X'.
WRITE: / 'In Subroutine S2, P1 =', P1.
*stop statement terminates the subroutine and

“Control goes to the end-of-selection event
STOP.
ENDFORM.
```

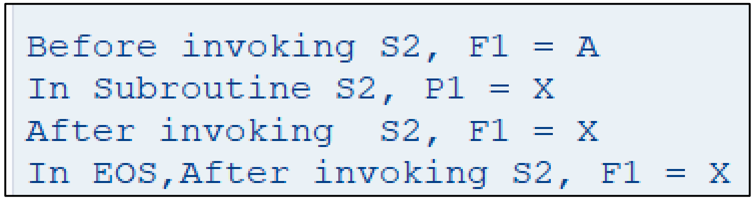
Demo: Subroutine – Pass By Value and Result

Code – without stop in subroutine



```
REPORT Z.
DATA: F1 TYPE C VALUE 'A'.
WRITE: / 'Before invoking S2, F1 =', F1.
*PERFORM S2 USING F1.
PERFORM S2 CHANGING F1.
WRITE: / 'After invoking  S2, F1 =', F1.

END-OF-SELECTION.
WRITE: / 'In EOS,After invoking S2, F1 =', F1.

FORM S2 CHANGING VALUE(P1).
*Above is pass by value and result
P1 = 'X'.
WRITE: / 'In Subroutine S2, P1 =', P1.
ENDFORM.
```

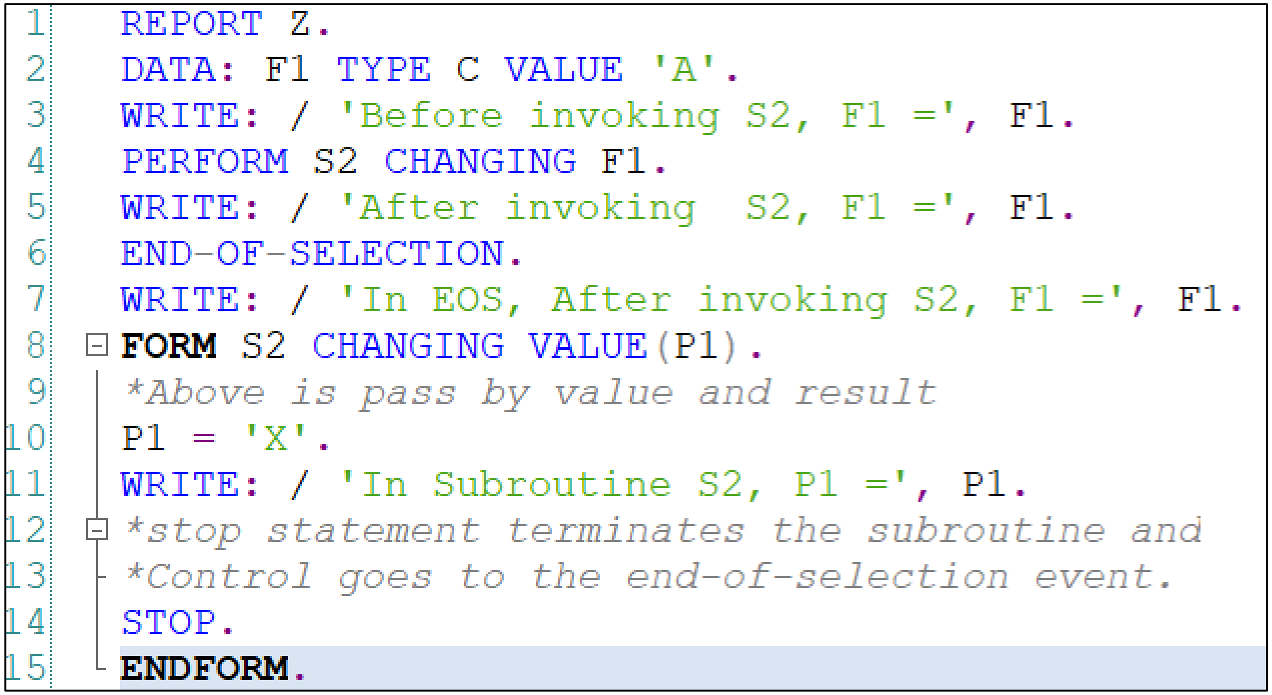
Output after execution



```
Before invoking S2, F1 = A
In Subroutine S2, P1 = X
After invoking  S2, F1 = X
In EOS,After invoking S2, F1 = X
```

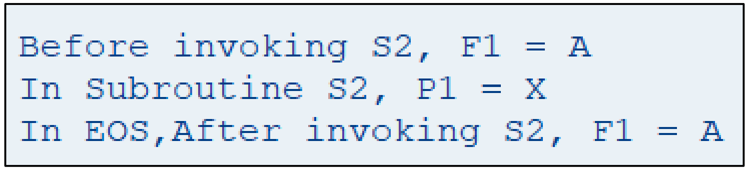
Demo: Subroutine – Pass By Value and Result

Code – with stop in subroutine



```
REPORT Z.
DATA: F1 TYPE C VALUE 'A'.
WRITE: / 'Before invoking S2, F1 =', F1.
PERFORM S2 CHANGING F1.
WRITE: / 'After invoking  S2, F1 =', F1.
END-OF-SELECTION.
WRITE: / 'In EOS, After invoking S2, F1 =', F1.
FORM S2 CHANGING VALUE(P1).
*Above is pass by value and result
P1 = 'X'.
WRITE: / 'In Subroutine S2, P1 =', P1.
*stop statement terminates the subroutine and
*Control goes to the end-of-selection event.
STOP.
ENDFORM.
```

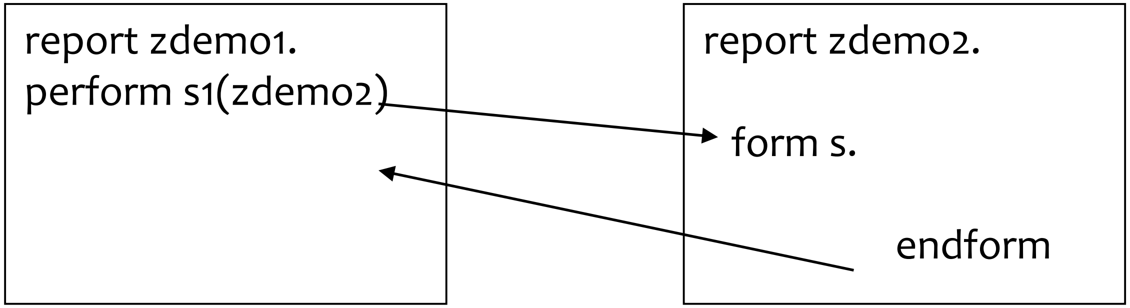
Output after execution



```
Before invoking S2, F1 = A
In Subroutine S2, P1 = X
In EOS,After invoking S2, F1 = A
```

Defining and Calling External Subroutines

An external subroutine is one that resides in a different program than the perform statement that calls it.
The below figure illustrates an external subroutine.



report zdemo1.
perform s1(zdemo2)

report zdemo2.

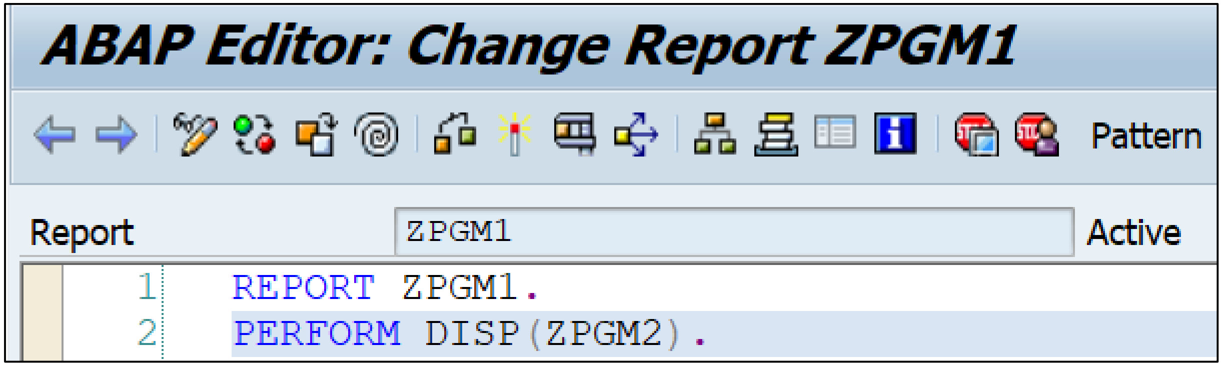
form s.

endform

When a perform calls an external subroutine

- The external program containing the subroutine is loaded
- The entire external program is checked for syntax
- Control transfers to the form in the external program
- The statements within the external subroutine are executed
- The endform transfers control back to the statement following the perform

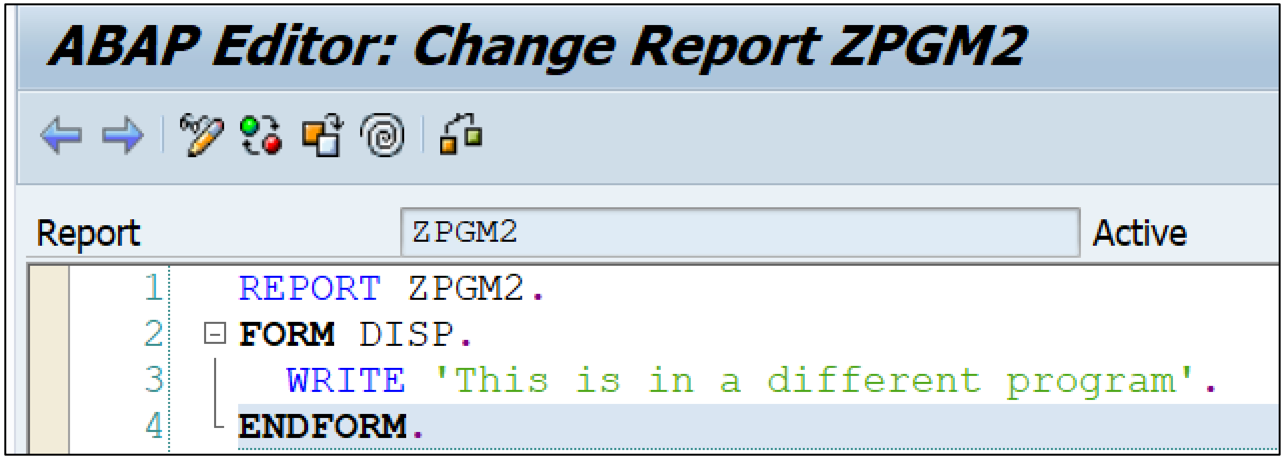
Demo1: Call subroutine defined in other program



ABAP Editor: Change Report ZPGM1

Report ZPGM1 Active

```
REPORT ZPGM1.
PERFORM DISP(ZPGM2).
```

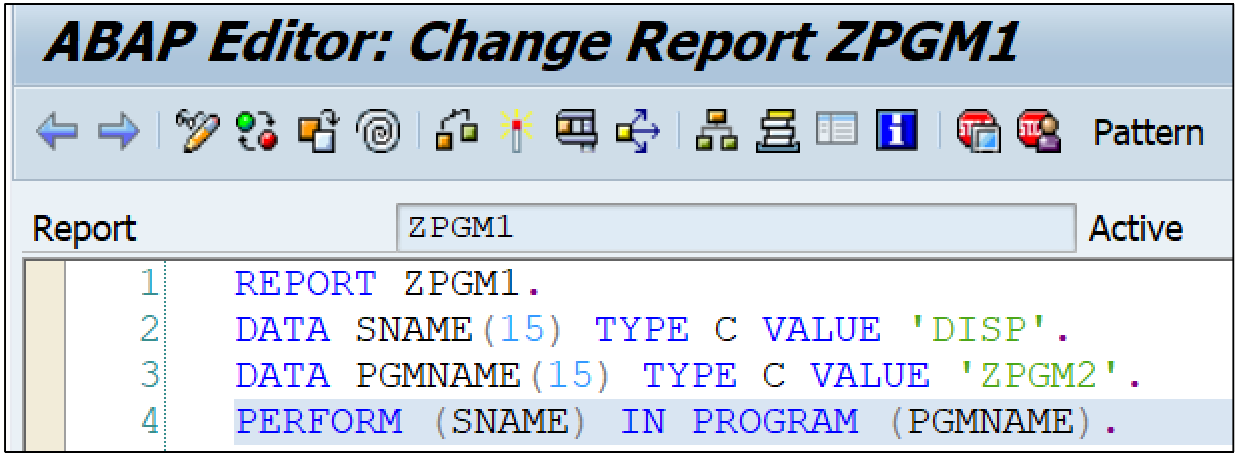


ABAP Editor: Change Report ZPGM2

Report ZPGM2 Active

```
REPORT ZPGM2.
FORM DISP.
  WRITE 'This is in a different program'.
ENDFORM.
```

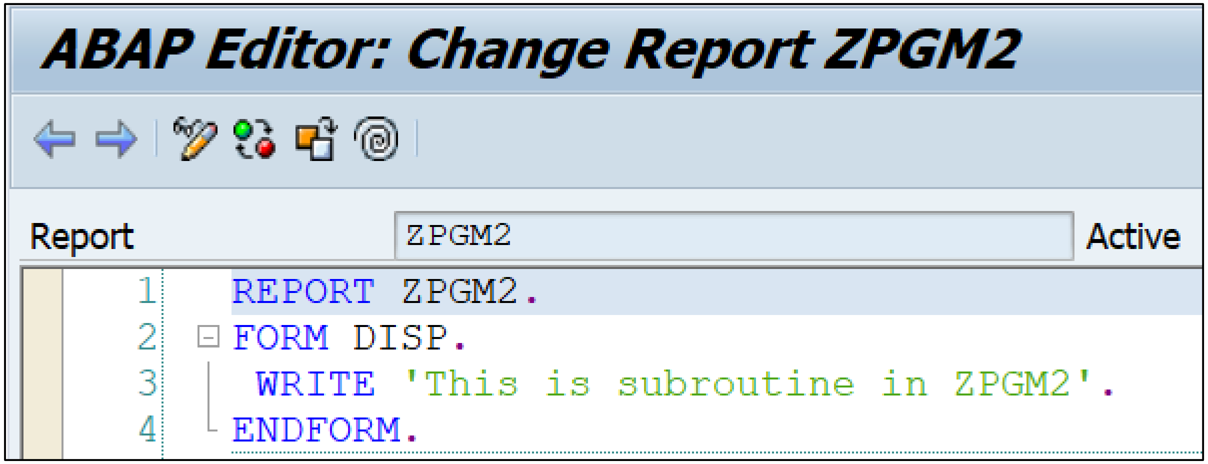
Demo2: Call subroutine defined in other program



ABAP Editor: Change Report ZPGM1

Report ZPGM1 Active

```
REPORT ZPGM1.
DATA SNAME(15) TYPE C VALUE 'DISP'.
DATA PGMNAME(15) TYPE C VALUE 'ZPGM2'.
PERFORM (SNAME) IN PROGRAM (PGMNAME).
```



ABAP Editor: Change Report ZPGM2

Report ZPGM2 Active

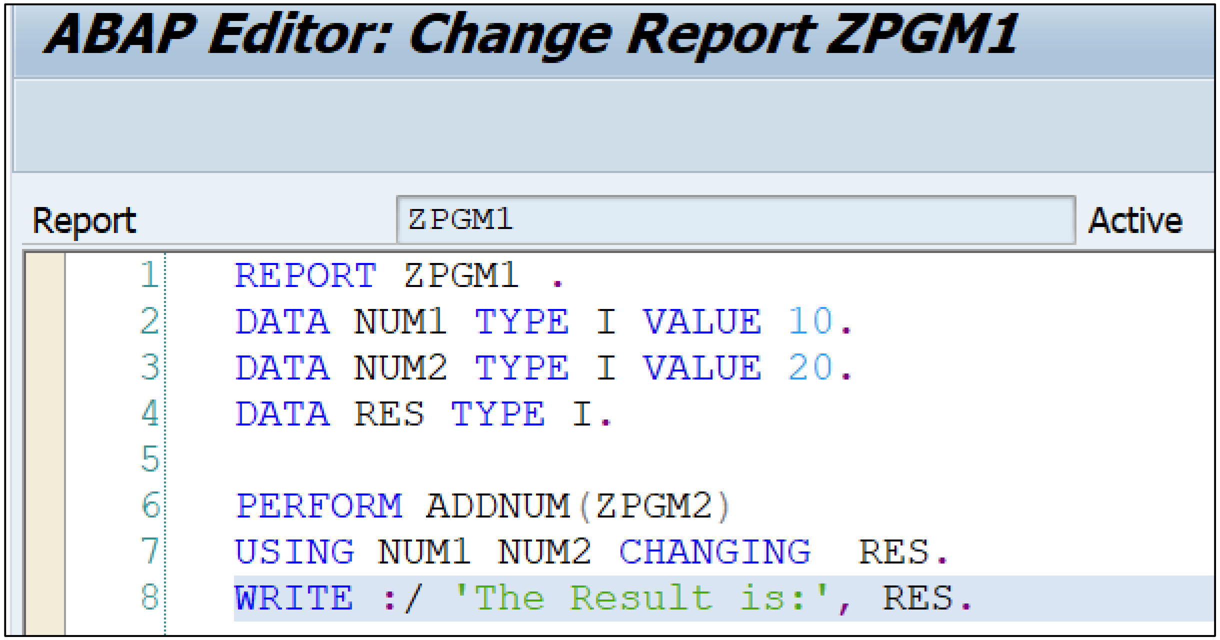
```
REPORT ZPGM2.
FORM DISP.
 WRITE 'This is subroutine in ZPGM2'.
ENDFORM.
```

Defining and Calling External Subroutines

External subroutines are very similar to internal subroutines:

- Both allow parameters to be passed
- Both allow typed formal parameters
- Both allow parameters to be passed by value, by value and result, and by reference.
- Both allow local variable definitions

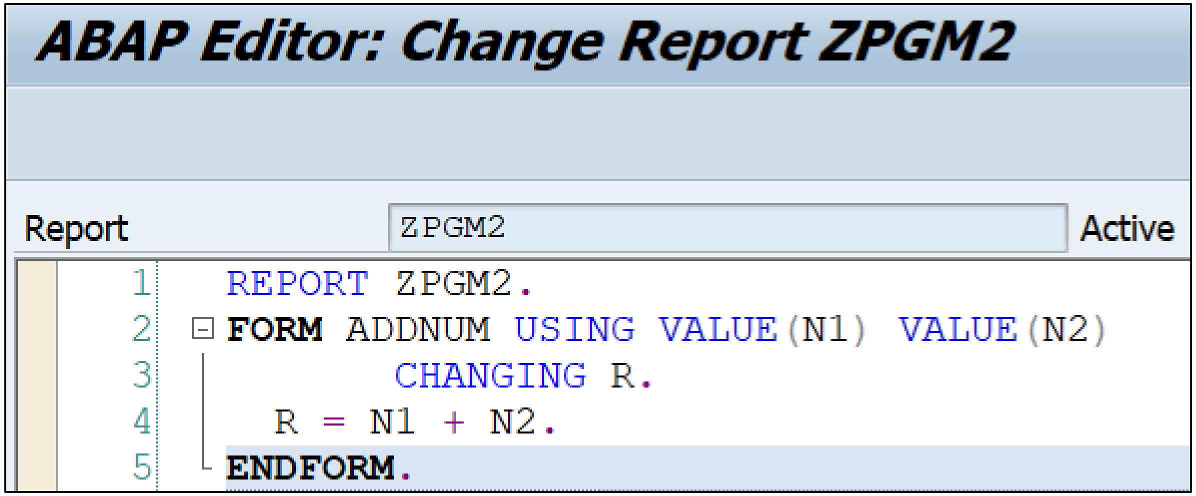
Demo3: Call subroutine defined in other program



ABAP Editor: Change Report ZPGM1

Report ZPGM1 Active

```
REPORT ZPGM1 .
DATA NUM1 TYPE I VALUE 10.
DATA NUM2 TYPE I VALUE 20.
DATA RES TYPE I.

PERFORM ADDNUM(ZPGM2)
USING NUM1 NUM2 CHANGING  RES.
WRITE :/ 'The Result is:', RES.
```



ABAP Editor: Change Report ZPGM2

Report ZPGM2 Active

```
REPORT ZPGM2.
FORM ADDNUM USING VALUE(N1) VALUE(N2)
       CHANGING R.
  R = N1 + N2.
ENDFORM.
```

Passing Internal Tables as Parameters to Subroutines

Internal tables can be passed as parameters to subroutines.

Passing an Internal Table without Header Line and automatically creating a header line in Subroutine

If the internal table doesn’t have a header line and you want to pass the body and create a header line in the subroutine, you can also use the syntax

- *form s1 tables it*.

This passes the body by reference, and creates a header line locally in the subroutine.

Changes made to the body of the internal table within the subroutine are immediately reflected in the original.

Demo: Pass an Internal table without Header Line to a Subroutine

Passing an Internal Table without Header Line – without automatically creating header line in subroutine

If an internal table doesn’t have a header line and you want to pass the body without automatically creating a header line in the subroutine, you can use the following syntax:

- form s1 using value(it)
- form s1 using it
- form s1 changing it
- form s1 changing value(it)

Demo: Pass an Internal table without Header Line to a Subroutine

The LOCAL statement

If you use the LOCAL statement when defining a subroutine, the system stores the current value of the field, field string, or field symbol you specify as soon as it enters the subroutine.

When you leave the routine, it restores the original value.

You can use LOCAL only after a FORM statement

Demo: Use LOCAL statement in subroutine