Debugger Tutorial

This tutorial introduces the basic features of the debugger and discusses different debugging methods. It takes you through a simple scenario that demonstrates how the debugger can be used to analyze runtime errors and control program execution.

It is important that you work through Sessions 1 to 5 in sequence.

Notes:

1. For ease of use, the tutorial primarily quotes direct commands to demonstrate the debugger features and not the alternative menu functions.
2. For a full description of all debugger features mentioned in this tutorial, refer to the relevant sections in the remainder of the *Debugger* documentation.

Prerequisites

- You should be familiar with programming in Natural.
- Before you start with Session 1, you need to create all example programs (DEBUG1P and DEBUG2P) and subprograms (DEBUG1N, DEBUG2N, DEBUG3N and DEBUG4N) provided in the section *Example Sources* later in this tutorial. Save and catalog these objects with the system command `STOW`.

Fundamentals of Debugging

The debugger can be used to interrupt the execution flow of a Natural object at a particular debug event and obtain information on the current status of the interrupted object such as the next statement to be executed, the value of a variable and the hierarchy (program levels) of objects called.

You basically need to take the following two major steps to pass control to the debugger for program interruption:

1. Activate the debugger with the system command `TEST ON`.

   This allows the debugger to receive control for each statement to be executed by the Natural runtime system.

2. Set one or more debug entries (breakpoints and watchpoints) for the Natural objects to be executed.

   This allows the debugger to decide when to take over control from the Natural runtime system and interrupt the program execution.

   A Natural error always interrupts the program execution. No debug entry is required then, the debugger steps in automatically.

The following is an overview of all possible program interruptions:

Program Interruption	**Explanation**
Breakpoint	Causes a program interruption for a statement line in a Natural object. The debugger interrupts the program execution whenever the statement line for which a breakpoint is set is to be executed, that is, *before* the statement contained in this line is processed.
Watchpoint	Causes a program interruption for a variable in a Natural object. The debugger interrupts the program execution whenever the contents of the variable for which a watchpoint is set have changed, that is, *after* the statement that references this variable is processed.
Step mode	Steps through the object during the program execution. Step mode is initiated by a debugger command and requires that the debugger previously received control because of a breakpoint or a watchpoint. In step mode, the debugger interrupts the program execution *before* each executable statement contained in this object is processed.
Natural error	Causes an automatic program interruption.

Session 1 - Analyzing a Natural Error

This session describes investigation methods for a Natural error that occurs during program execution.

To simulate a Natural error

1. From the NEXT prompt, execute DEBUG1P.

   The following Natural error message appears: `DEBUG1N 0180 NAT0954 Abnormal termination S0C7 during program execution.`

The message points to line 180 in the subprogram DEBUG1N: `BONUS := SALARY * PERCENT / 100`. This indicates that incorrect values are returned for one or more of the variables referenced. However, at this point, this is no clear evidence of what actually causes the problem; and it could be difficult to determine the cause if the variable values were retrieved from a database (as is typical for employee records).

To activate the debugger for further problem investigation

1. At the NEXT prompt, enter the following:

   ```
   TEST ON
   ```

   The message `Test mode started.` indicates that the debugger is activated.

   Note:
   `TEST ON` remains active for the duration of the current session or until you enter `TEST OFF` to deactivate the debugger.

2. Again, execute DEBUG1P from the NEXT prompt.

   A **Debug Break** window similar to the example below appears:

   ```
   +------------------- Debug Break -------------------+
   | Break by ABEND S0C7 at NATARI2+2A4-4 (NAT0954)    |
   | at line  180 in subprogram DEBUG1N (level 2)      |
   | in library DEBUG    in system file (10,32).       |
   |                                                   |
   |         G   Go                                    |
   |         L   List break                            |
   |         M   Debug Main Menu                       |
   |         N   Next break command                    |
   |         R   Run (set test mode OFF)               |
   |         S   Step mode                             |
   |         V   Variable maintenance                  |
   |                                                   |
   | Code .. G                                         |
   |                                                   |
   | Abnormal termination S0C7 during program execution|
   | PF2=Step,PF13=Next,PF14=Go,PF15=Menu,PF17=SkipS   |
   +---------------------------------------------------+
   ```

   Since a Natural error occurs, the debugger steps in automatically and displays the **Debug Break** window.

   Additional information on where the error occurs is displayed at the top of the window: the module (`NATARI2`) in the Natural nucleus (helpful for Software AG technical support), the type of object (`subprogram`) the library (`DEBUG`) and the database ID and file number (`10,32`) of the system file.

   The **Debug Break** window also provides debugger functions that can be used, for example, to continue the program execution (**Go** or **Run**), invoke the debugger maintenance menu (**Debug Main Menu**) or activate step mode. You execute a function by using either the appropriate function code or PF key.

To inspect the erroneous statement line

1. In the **Code** field, replace the default entry `G` by `L` to execute the **List break** function.

   The source of DEBUG1N is displayed:

```
13:48:54              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON                 - List Object Source -            Object DEBUG1N
                                                                Bottom of data
Co Line Source                                                   Message
__ 0070   2 NUMCHILD  (N2)                                      |
__ 0080   2 ENTRYDATE (D)                                       |
__ 0090   2 SALARY    (P7.2)                                    |
__ 0100   2 BONUS     (P7.2)                                    |
__ 0110 LOCAL                                                   |
__ 0120 1 TARGETDATE  (D)    INIT <D'2009-01-01'>               |
__ 0130 1 DIFFERENCE  (P3.2)                                    |
__ 0140 1 PERCENT     (P2.2) INIT <3.5>                         |
__ 0150 END-DEFINE                                              |
__ 0160 DIFFERENCE := (TARGETDATE - ENTRYDATE) / 365            |
__ 0170 IF DIFFERENCE GE 10        /* BONUS FOR YEARS IN COMPAN | last line
__ 0180   BONUS := SALARY * PERCENT / 100                       | * NAT0954 *
__ 0190 END-IF                                                  |
__ 0200 SALARY := SALARY + 1800    /* SALARY PLUS ANNUAL INCREA |
__ 0210 END                                                     |

Command ===>



Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Scan  Flip  -     +     Li Br <     >     Canc
```

   `last line` indicates that the statement contained in line 170 is the last statement that executed successfully.

   The statement in line 180 which causes the problem is highlighted and annotated with `* NAT0954 *`.

   This indicates that the error is caused by either the contents of the variable `SALARY` or `PERCENT`. Most likely, this is `SALARY` since `PERCENT` is properly initialized.

To check the contents of SALARY

1. In the Command line, enter the following:

```
DIS VAR SALARY
```

   A **Display Variable** screen similar to the example below appears for the variable `SALARY`:

```
18:59:51              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON          - Display Variable (Alphanumeric) -      Object DEBUG1N


Name ...... EMPLOYEE.SALARY
Fmt/Len ... P 7.2
Type ...... parameter
Index .....
Range .....

Position ..
Contents ..


Command ===>


Variable contains invalid data.

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Mod   Flip              Li Br Alpha Hex   Canc
```

The message `Variable contains invalid data.` indicates that the contents of the variable, which seems to be blank, does not match the format of the variable. This becomes clear when you view the hexadecimal representation of the variable contents as described in the next step.

2. Press PF11 (Hex) to display the hexadecimal contents of the variable.

   The screen now looks similar to the example below:

```
11:13:33              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON          - Display Variable (Hexadecimal) -       Object DEBUG1N


Name ...... EMPLOYEE.SALARY
Fmt/Len ... P 7.2
Type ...... parameter
Index .....
Range .....

Position ..
Contents .. 4040404040


Command ===>



Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Mod   Flip              Li Br Alpha Hex   Canc
```

The hexadecimal value shows that the variable is not in packed numeric format, thus leading to a calculation error during the program execution. DEBUG1P obviously provides DEBUG1N with an incorrect value for SALARY.

Tip:
You can press PF10 (Alpha) to switch back to the alphanumeric representation.

3. In the Command line, enter the following:

```
GO
```

The command `GO` returns control from the debugger to the Natural runtime system, which continues the program execution until the end of the program or the next debug event. In this case, there is no additional debug event and the NEXT prompt appears with the known Natural error message.

To correct `SALARY` in the object source

1. Open DEBUG1P with the program editor and remove the comment sign (*) entered for `SALARY := 99000`.

2. Save and catalog the program with the system command `STOW`.

3. Execute DEBUG1P.

   The debugger does not interrupt the program though `TEST ON` is still set. The program executes successfully and outputs a report:

```
Page      1                                               07-09-06  15:28:06

EMPLOYEE RECEIVES:   100800.00
    PLUS BONUS OF:     3465.00





NEXT                                                            LIB=DEBUG
```

Session 2 - Using a Breakpoint

You can interrupt the program execution at a specific statement line by setting a breakpoint for this line.

To set a breakpoint for a statement line in DEBUG1N

1. At the NEXT prompt, enter the following:

```
TEST SET BP DEBUG1N 170
```

The message `Breakpoint DEBUG1N0170 set at line 170 of object DEBUG1N.` confirms that a breakpoint with the name `DEBUG1N0170` is set for statement line 170 in the DEBUG1N subprogram.

Notes:

1. A breakpoint can only be set for an executable statement. If you try to set a statement for a non-executable statement, an appropriate error message appears.
2. A breakpoint is usually only valid during the current Natural session. If required, you can save a

breakpoint for future sessions: see *Saving Breakpoints and Watchpoints* in *Additional Hints for Using the Debugger*.

2. Execute DEBUG1P.

   The debugger now interrupts the program execution at the statement line, where the new breakpoint is set. The **Debug Break** window appears:

```
+------------------- Debug Break -------------------+
| Break by breakpoint DEBUG1N0170                   |
| at line  170 in subprogram DEBUG1N (level 2)      |
| in library DEBUG    in system file (10,32).       |
|                                                   |
|        G   Go                                     |
|        L   List break                             |
|        M   Debug Main Menu                        |
|        N   Next break command                     |
|        R   Run (set test mode OFF)                |
|        S   Step mode                              |
|        V   Variable maintenance                   |
|                                                   |
| Code .. G                                         |
|                                                   |
|                                                   |
| PF2=Step,PF13=Next,PF14=Go,PF15=Menu,PF17=SkipS   |
+---------------------------------------------------+
```

   The window indicates the name of the breakpoint, the corresponding statement line and object and the library that contains the object. It also indicates the operational level of subprogram DEBUG1N.

To view the statement indicated in the Debug Break window

- Execute the **List break** function.

  The source of DEBUG1N is displayed on the **List Object Source** screen:

```
11:36:45              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON                - List Object Source -           Object DEBUG1N
                                                              Bottom of data
Co Line Source                                                 Message
__ 0070   2 NUMCHILD  (N2)                                    |
__ 0080   2 ENTRYDATE (D)                                     |
__ 0090   2 SALARY    (P7.2)                                  |
__ 0100   2 BONUS     (P7.2)                                  |
__ 0110 LOCAL                                                 |
__ 0120 1 TARGETDATE  (D)    INIT <D'2009-01-01'>             |
__ 0130 1 DIFFERENCE  (P3.2)                                  |
__ 0140 1 PERCENT     (P2.2) INIT <3.5>                       |
__ 0150 END-DEFINE                                            |
__ 0160 DIFFERENCE := (TARGETDATE - ENTRYDATE) / 365          | last line
__ 0170 IF DIFFERENCE GE 10        /* BONUS FOR YEARS IN COMPAN | DEBUG1N0170
__ 0180   BONUS := SALARY * PERCENT / 100                     |
__ 0190 END-IF                                                |
__ 0200 SALARY := SALARY + 1800    /* SALARY PLUS ANNUAL INCREA |
__ 0210 END                                                   |

Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Scan  Flip  -     +     Li Br <     >     Canc
```

Statement line 170 indicated in the **Debug Break** window is highlighted. The **Message** column indicates the name of the breakpoint (`DEBUG1N0170`) set for this statement line and the last statement line executed (line 160 as indicated by `last line`).
Remember: A breakpoint interrupts the program execution *before* the statement for which the breakpoint is set is processed.

There are several direct commands you can enter on the **List Object Source** screen to obtain more information on the current object. As an example, you can view all variables as described in the following step.

To display a list of variables contained in DEBUG1N

1. In the Command line, enter the following:

```
DIS VAR
```

A **Display Variables** screen similar to the example below appears:

```
11:06:13              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON         - Display Variables (Alphanumeric) -       Object DEBUG1N
                                                                           All
Co Le Variable Name                   F       Leng Contents               Msg.
    1 EMPLOYEE
__  2 NAME                            A         20 MEIER
__  2 ENTRYDATE                       D            1989-01-01
__  2 SALARY                          P        7.2 99000.00
__  2 BONUS                           P        7.2 *** invalid data ***
__  1 TARGETDATE                      D            2009-01-01
__  1 DIFFERENCE                      P        3.2 20.00
__  1 PERCENT                         P        2.2 3.50




Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Zoom  Flip  -     +     Li Br Alpha Hex   Canc
```

The screen lists all variables defined in DEBUG1N. You can neglect the remark `invalid data` for `BONUS`. In this case, it is not essential whether `BONUS` is properly initialized since it is used as a target operand only. However, to exercise another debugger command, change the contents of `BONUS` in the following step.

To check and modify the contents of `BONUS`

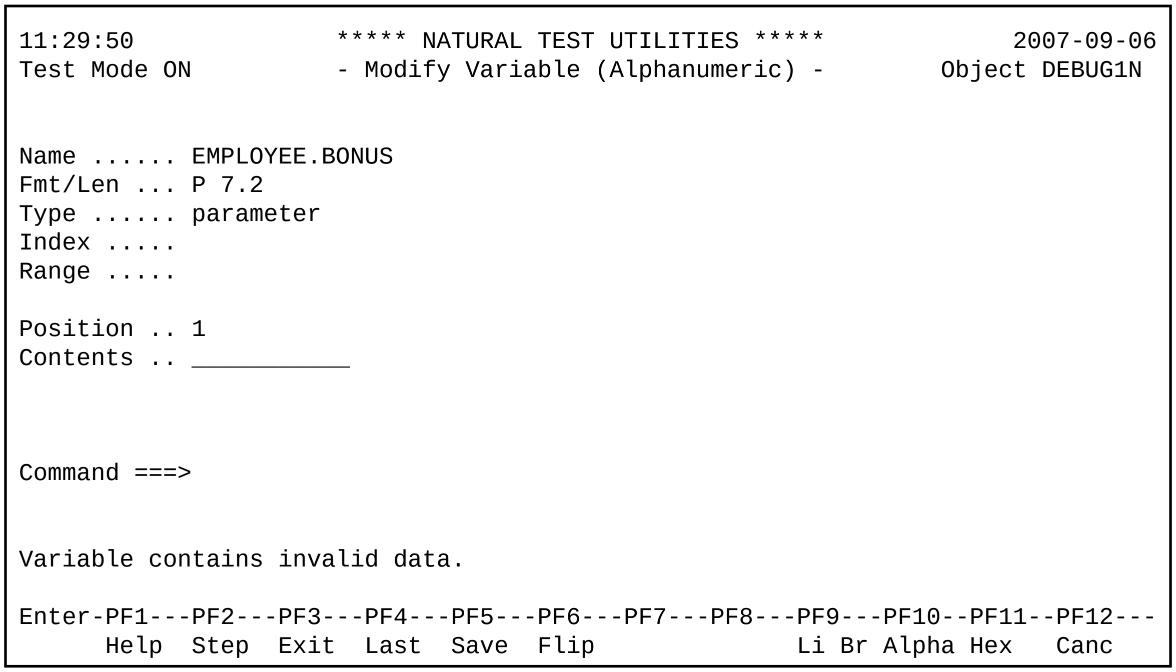
1. In the **Co** column, next to `BONUS`, enter the following:

```
MO
```

Or:
In the Command line, enter the following:

```
MOD VAR BONUS
```

A **Modify Variable** screen similar to the example below appears:

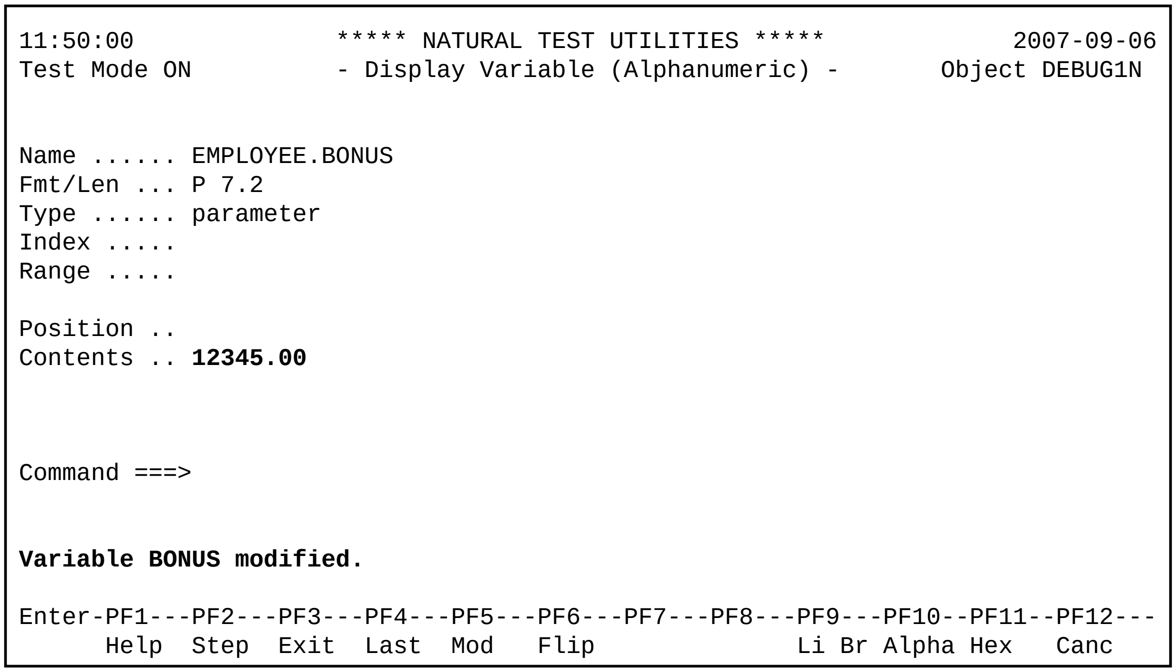


```
11:29:50             ***** NATURAL TEST UTILITIES *****             2007-09-06
Test Mode ON         - Modify Variable (Alphanumeric) -        Object DEBUG1N


Name ...... EMPLOYEE.BONUS
Fmt/Len ... P 7.2
Type ...... parameter
Index .....
Range .....

Position .. 1
Contents .. ___________



Command ===>


Variable contains invalid data.

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Save  Flip              Li Br Alpha Hex   Canc
```

2. You can use the hexadecimal display to verify that the variable is not in packed numeric format. Press PF10 (Alpha) to switch back to the alphanumeric representation.

3. In the **Contents** field, enter a value in packed numeric format, for example, `12345.00` and press PF5 (Save).

   The screen now looks similar to the example below:



```
11:50:00             ***** NATURAL TEST UTILITIES *****             2007-09-06
Test Mode ON         - Display Variable (Alphanumeric) -       Object DEBUG1N


Name ...... EMPLOYEE.BONUS
Fmt/Len ... P 7.2
Type ...... parameter
Index .....
Range .....

Position ..
Contents .. 12345.00



Command ===>


Variable BONUS modified.

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Mod   Flip              Li Br Alpha Hex   Canc
```

A message confirms the modification of **Contents**.

4. Press PF9 (Li Br) or PF3 (Exit).

   The **List Object Source** screen appears.

5. In the Command line, enter the following:

   ```
   GO
   ```

   The debugger returns control to the Natural runtime system, which finishes executing DEBUG1P since no further debug event occurs. The report produced by the program is output:

   ```
   Page      1                                                07-09-06  10:02:51

   EMPLOYEE RECEIVES:   100800.00
       PLUS BONUS OF:     3465.00




   NEXT                                                               LIB=DEBUG
   ```

6. Before you continue with the next session, delete all current breakpoints by entering the following at the NEXT prompt:

   ```
   TEST DEL BP * *
   ```

   A message appears confirming that all breakpoint (in this case, only one breakpoint) are deleted.

Session 3 - Using a Watchpoint

DEBUG1P and DEBUG1N perform a calculation for a single employee’s bonus and salary payment. If multiple employee records were processed, you would probably test whether the variable `BONUS` is now updated correctly. This is done by setting a watchpoint for this variable. A watchpoint allows the debugger to interrupt the program execution when the contents of the specified variable change.

To set a watchpoint for the variable `BONUS`

1. At the NEXT prompt, enter the following:

   ```
   TEST SET WP DEBUG1N BONUS
   ```

   The message `Watchpoint BONUS set for variable EMPLOYEE.BONUS.` confirms that a watchpoint is set for the variable `BONUS` in the DEBUG1N example subprogram.

   Notes:

1. If you enter a debugger direct command in the Command line of a debugger screen, you must omit the keyword `TEST`. For example, instead of `TEST SET WP DEBUG1N BONUS`, you would then enter `SET WP DEBUG1N BONUS` only.
2. A watchpoint is usually only valid during the current Natural session. If required, you can save a watchpoint for future sessions: see *Saving Breakpoints and Watchpoints* in *Additional Hints for Using the Debugger*.

2. Execute DEBUG1P from the NEXT prompt.

   The debugger interrupts the program execution at the new watchpoint and invokes the **Debug Break** window:

```
+------------------- Debug Break -------------------+
| Break by watchpoint BONUS                         |
| at line  180 in subprogram DEBUG1N (level 2)      |
| in library DEBUG    in system file (10,32).       |
|                                                   |
|         G   Go                                    |
|         L   List break                            |
|         M   Debug Main Menu                       |
|         N   Next break command                    |
|         R   Run (set test mode OFF)               |
|         S   Step mode                             |
|         V   Variable maintenance                  |
|                                                   |
| Code .. G                                         |
|                                                   |
|                                                   |
| PF2=Step,PF13=Next,PF14=Go,PF15=Menu,PF17=SkipS   |
+---------------------------------------------------+
```

   The window indicates that a watchpoint was detected in line 180. This line contains the statement that processes the variable `BONUS`.

   The debugger interrupted the program execution *after* the statement for `BONUS` was processed. Only then could the debugger recognize that the contents of the variable had changed.

3. Execute the **List break** function.

   The **List Object Source** now looks similar to the example below:

```
16:24:46              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON                - List Object Source -            Object DEBUG1N
                                                               Bottom of data
Co Line Source                                                  Message
__ 0070   2 NUMCHILD  (N2)                                    |
__ 0080   2 ENTRYDATE (D)                                     |
__ 0090   2 SALARY    (P7.2)                                  |
__ 0100   2 BONUS     (P7.2)                                  |
__ 0110 LOCAL                                                 |
__ 0120 1 TARGETDATE  (D)    INIT <D'2009-01-01'>             |
__ 0130 1 DIFFERENCE  (P3.2)                                  |
__ 0140 1 PERCENT     (P2.2) INIT <3.5>                       |
__ 0150 END-DEFINE                                            |
__ 0160 DIFFERENCE := (TARGETDATE - ENTRYDATE) / 365          |
__ 0170 IF DIFFERENCE GE 10        /* BONUS FOR YEARS IN COMPAN | DEBUG1N0170
__ 0180   BONUS := SALARY * PERCENT / 100                     | BONUS
__ 0190 END-IF                                                |
__ 0200 SALARY := SALARY + 1800    /* SALARY PLUS ANNUAL INCREA |
__ 0210 END                                                   |

Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Scan  Flip  -     +     Li Br <     >     Canc
```

The statement which references the variable BONUS is highlighted and the **Message** column indicates the name of the watchpoint set for the variable.

To check for changes in BONUS

1. In the Command line, enter the following:

```
DIS VAR BONUS
```

The **Display Variable** screen appears and displays a value of 3465.00 in the **Contents** field. This shows that the contents of the variable BONUS have changed.

2. Press PF3 (Exit) to return to the **List Object Source** screen.

To check for changes in SALARY

1. To test the contents of the variable SALARY in a later step, set a breakpoint for SALARY by entering the following in the **Co** column of line 200:

```
SE
```

From the **List Object Source** screen, a line command such as SE is a convenient alternative to using the SET BP direct command.

The **Message** column indicates that a breakpoint (BP) is set for line 200:

```
17:55:58              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON                 - List Object Source -            Object DEBUG1N
                                                                Bottom of data
Co Line Source                                                   Message
__ 0070   2 NUMCHILD  (N2)                                     |
__ 0080   2 ENTRYDATE (D)                                      |
__ 0090   2 SALARY    (P7.2)                                   |
__ 0100   2 BONUS     (P7.2)                                   |
__ 0110 LOCAL                                                  |
__ 0120 1 TARGETDATE  (D)    INIT <D'2009-01-01'>              |
__ 0130 1 DIFFERENCE  (P3.2)                                   |
__ 0140 1 PERCENT     (P2.2) INIT <3.5>                        |
__ 0150 END-DEFINE                                             |
__ 0160 DIFFERENCE := (TARGETDATE - ENTRYDATE) / 365           |
__ 0170 IF DIFFERENCE GE 10        /* BONUS FOR YEARS IN COMPAN | DEBUG1N0170
__ 0180   BONUS := SALARY * PERCENT / 100                      | BONUS
__ 0190 END-IF                                                 |
__ 0200 SALARY := SALARY + 1800    /* SALARY PLUS ANNUAL INCREA | BP set
__ 0210 END                                                    |

Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Scan  Flip  -     +     Li Br <     >     Canc
```

2. In the Command line, enter the following:

```
GO
```

The **Debug Break** window appears:

```
+------------------- Debug Break -------------------+
| Break by breakpoint DEBUG1N0200                   |
| at line  200 in subprogram DEBUG1N (level 2)      |
| in library DEBUG    in system file (10,32).       |
|                                                   |
|         G   Go                                    |
|         L   List break                            |
|         M   Debug Main Menu                       |
|         N   Next break command                    |
|         R   Run (set test mode OFF)               |
|         S   Step mode                             |
|         V   Variable maintenance                  |
|                                                   |
| Code .. G                                         |
|                                                   |
|                                                   |
| PF2=Step,PF13=Next,PF14=Go,PF15=Menu,PF17=SkipS   |
+---------------------------------------------------+
```

3. Execute the **List break** function.

The **List Object Source** screen now looks similar to the example below:

```
10:49:31              ***** NATURAL TEST UTILITIES *****             2007-09-06
Test Mode ON                - List Object Source -              Object DEBUG1N
                                                                 Bottom of data
Co Line Source                                                    Message
__ 0070   2 NUMCHILD  (N2)                                      |
__ 0080   2 ENTRYDATE (D)                                       |
__ 0090   2 SALARY    (P7.2)                                    |
__ 0100   2 BONUS     (P7.2)                                    |
__ 0110 LOCAL                                                   |
__ 0120 1 TARGETDATE  (D)    INIT <D’2009-01-01’>               |
__ 0130 1 DIFFERENCE  (P3.2)                                    |
__ 0140 1 PERCENT     (P2.2) INIT <3.5>                         |
__ 0150 END-DEFINE                                              |
__ 0160 DIFFERENCE := (TARGETDATE - ENTRYDATE) / 365            |
__ 0170 IF DIFFERENCE GE 10        /* BONUS FOR YEARS IN COMPAN | DEBUG1N0170
__ 0180   BONUS := SALARY * PERCENT / 100                       | last line
__ 0190 END-IF                                                  |
__ 0200 SALARY := SALARY + 1800    /* SALARY PLUS ANNUAL INCREA | DEBUG1N0200
__ 0210 END                                                     |

Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Scan  Flip  -     +     Li Br <     >     Canc
```

Since this is a breakpoint, the statement that references (and updates) SALARY has not yet been executed. As a result, the contents of the variable have not changed.

4. In the Command line, enter `DIS VAR SALARY` to verify that the contents of SALARY are unchanged.

   The variable screen proves that SALARY still contains 99000, the initial value assigned in DEBUG1P.

5. To view the update of the variable contents, step to the next statement by choosing either of the following methods:

   In the Command line, enter the following:

   ```
   STEP
   ```

   Or:
   Press PF2 (Step).

   The screen now looks similar to the example below:

```
13:38:24              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON                - List Object Source -            Object DEBUG1N
                                                               Bottom of data
Co Line Source                                                  Message
__ 0070   2 NUMCHILD  (N2)                                    |
__ 0080   2 ENTRYDATE (D)                                     |
__ 0090   2 SALARY    (P7.2)                                  |
__ 0100   2 BONUS     (P7.2)                                  |
__ 0110 LOCAL                                                 |
__ 0120 1 TARGETDATE  (D)    INIT <D'2009-01-01'>             |
__ 0130 1 DIFFERENCE  (P3.2)                                  |
__ 0140 1 PERCENT     (P2.2) INIT <3.5>                       |
__ 0150 END-DEFINE                                            |
__ 0160 DIFFERENCE := (TARGETDATE - ENTRYDATE) / 365          |
__ 0170 IF DIFFERENCE GE 10        /* BONUS FOR YEARS IN COMPAN | DEBUG1N0170
__ 0180   BONUS := SALARY * PERCENT / 100                     |
__ 0190 END-IF                                                |
__ 0200 SALARY := SALARY + 1800    /* SALARY PLUS ANNUAL INCREA | last line
__ 0210 END                                                   | step mode

Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Scan  Flip  -     +     Li Br <     >     Canc
```

You skipped one line and processed the next executable statement in line 200, which updates `SALARY`. The **Message** column indicates that step mode is set. In step mode, the debugger continues the program execution at the next executable statement.

6. In the Command line, enter `DIS VAR SALARY` to check the variable contents.

   The **Display Variable** screen appears and displays a value of `100800.00` in the **Contents** field. This proves that the contents of the variable `SALARY` have changed.

7. In the Command line, enter the following:

```
GO
```

The debugger returns control to the Natural runtime system, which finishes executing DEBUG1P since no further debug event occurs. The report produced by the program is output.

Session 4 - Tracing the Logical Flow of Programs

This session describes debugging methods you can use to better understand, overview and control a complex Natural application with numerous objects.

The session starts out with instructions for analyzing the logical flow of an application on the statement level. It then demonstrates how breakpoints can be used to find out the sequence in which programs are executed.

The instructions in this session are based on a simple (but sufficient for demonstration) example application that consists of one program (DEBUG2P) and three subprograms (DEBUG2N, DEBUG3N and DEBUG4N).

To set a breakpoint at program begin or end

1. Set a breakpoint for DEBUG2P by entering the following at the NEXT prompt:

   ```
   TEST SET BP DEBUG2P BEG
   ```

   The message `Breakpoint DEBUG2P-BEG set at line BEG of object DEBUG2P.` confirms that a breakpoint is set in DEBUG1N.

   Using the keyword `BEG` instead of a specific line number has the effect that the breakpoint is set at the beginning of the program, that is, for the first statement to be executed. This can even be the `DEFINE DATA` statement, for example, if an `INIT` clause is used, which generates an executable statement when the program is cataloged.

   Tip:

   You can also specify the keyword `END` to set a breakpoint for the last statement to be executed. This can be the `END` statement but also the `FETCH` or `CALLNAT` statement.

2. Execute DEBUG2P.

   The **Debug Break** window appears:

   ```
   +------------------- Debug Break -------------------+
   | Break by breakpoint DEBUG2P-BEG                   |
   | at line  130 in program DEBUG2P (level 1)         |
   | in library DEBUG    in system file (10,32).       |
   |                                                   |
   |         G   Go                                    |
   |         L   List break                            |
   |         M   Debug Main Menu                       |
   |         N   Next break command                    |
   |         R   Run (set test mode OFF)               |
   |         S   Step mode                             |
   |         V   Variable maintenance                  |
   |                                                   |
   | Code .. G                                         |
   |                                                   |
   |                                                   |
   | PF2=Step,PF13=Next,PF14=Go,PF15=Menu,PF17=SkipS   |
   +---------------------------------------------------+
   ```

   The debugger now steps in at the first breakpoint set for the program.

3. Execute the **List break** function to check the source and see that the debugger now steps in at the first executable statement `NAME := 'MEIER'`.

To step through an application

1. On the **List Object Source** screen, set step mode by either pressing PF2 (Step) or entering `STEP` in the Command line.

   The last statement executed is annotated with `last line`. The next statement to be executed is highlighted and annotated with `step mode`.

   Tip:

   If you do not want the debugger to pause at every single statement but step through an application more quickly, in the `STEP` command, specify the number of statements you want to skip, for example: `STEP 2` or `STEP 10`.

2. Press PF2 (Step) repeatedly until the `CALLNAT` statement is annotated with `step mode`.

3. Continue with PF2 (Step) and execute the `CALLNAT`.

   The invoked subprogram DEBUG2N is displayed, where the next statement to be executed is highlighted:

```
11:59:19              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON                - List Object Source -             Object DEBUG2N
                                                                   Top of data
Co Line Source                                                   Message
__ 0010 ** SUBPROGRAM DEBUG2N: CALLS 'DEBUG3N' AND 'DEBUG4N'FOR |
__ 0020 ******************************************************* |
__ 0030 DEFINE DATA                                             | step mode
__ 0040 PARAMETER                                               |
__ 0050 1 EMPLOYEE                                              |
__ 0060   2 NAME      (A20)                                     |
__ 0070   2 NUMCHILD  (N2)                                      |
__ 0080   2 ENTRYDATE (D)                                       |
__ 0090   2 SALARY    (P7.2)                                    |
__ 0100   2 BONUS     (P7.2)                                    |
__ 0110 LOCAL                                                   |
__ 0120 1 TARGETDATE  (D)    INIT <D'2009-01-01'>               |
__ 0130 1 DIFFERENCE  (P3.2)                                    |
__ 0140 1 PERCENT     (P2.2) INIT <3.5>                         |
__ 0150 END-DEFINE                                              |

Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Scan  Flip  -     +     Li Br <     >     Canc
```

   As an alternative, you could skip the `CALLNAT` by entering `STEP SKIP` in the Command line.

   You would then only step through the statements in the invoking program DEBUG2 but not through the statements within an invoked subprogram.

To view the levels at which the objects are executed

1. In the **List Object Source** screen of DEBUG2N, enter the following in the Command line:

   ```
   OBJCHAIN
   ```

   A **Break Information** screen similar to the example below appears:

   ```
   13:45:34               ***** NATURAL TEST UTILITIES *****            2007-09-06
                                - Break Information -

   No GDA active for the current program.

   Break by step mode
   at line   30 in subprogram DEBUG2N (level 2)
   in library DEBUG    in system file (10,32).
   ```

   In addition to the object information already known, this screen indicates whether the program references a GDA (global data area).

2. Press ENTER to scroll down one page.

   The screen now looks similar to the example below:

   ```
   13:46:34               ***** NATURAL TEST UTILITIES *****            2007-09-06
                              - Current Object Chain -

   Level Name     Type        Line Library   DBID   FNR
     2   DEBUG2N  Subprogram     0 DEBUG       10    32
     1   DEBUG2P  Program      170 DEBUG       10    32
   ```

   This screen indicates the operational levels at which the objects are executed: subprogram DEBUG2N is executed at level 2 and program DEBUG2P (which invokes the subprogram) is executed at the superior level 1.

3. Press ENTER.

   The **List Object Source** screen appears.

4. In the Command line, enter the following:

   ```
   GO
   ```

   The debugger returns control to the Natural runtime system, which finishes executing DEBUG2P since no further debug event occurs. The report produced by the program is output:

```
Page      1                                                07-09-06  10:04:21

EMPLOYEE RECEIVES:    99300.00
    PLUS BONUS OF:     3565.00




NEXT                                                          LIB=DEBUG
```

5. Delete all breakpoints currently set by entering the following at the NEXT prompt:

   ```
   TEST DEL BP * *
   ```

   A message appears confirming that all breakpoints are deleted.

To set breakpoints to follow the program execution

1. At the NEXT prompt, enter the following:

   ```
   TEST SET BP ALL BEG
   ```

   The message `Breakpoint ALL-BEG set at line BEG of object ALL.` appears.

   This indicates that you have set a breakpoint for the first executable statement of each object to be executed.

2. Execute DEBUG2P.

   A **Debug Break** window appears for DEBUG2P.

3. Execute the **Go** function repeatedly.

   Each time you execute **Go**, the next object invoked is indicated in the **Debug Break** window (DEBUG2N first and then DEBUG3N and DEBUG4N). Thus, you can easily determine which objects are invoked at what point during the program execution. Additionally, for each object, you can apply the menu functions of the **Debug Break** window.

4. When the NEXT prompt appears, delete all breakpoints currently set by entering the following:

   ```
   TEST DEL BP * *
   ```

   A message appears confirming that all breakpoints are deleted.

Session 5 - Using Statistics about the Program Execution

You can use the debugger to view statistical information on which objects are called and how often they are called. Additionally, you can find out which statements are executed, and how often.

To check what objects are called during program execution

1. At the NEXT prompt, enter the following:

   ```
   TEST SET CALL ON
   ```

   The message `Call statistics started.` confirms that the statistics function is activated.

2. Execute DEBUG2P.

   The debugger logs all object calls executed, and the report produced by the program is output.

3. At the NEXT prompt, enter the following:

   ```
   TEST DIS CALL
   ```

   A **Display Called Objects** screen similar to the example below appears:

   ```
   10:43:47              ***** NATURAL TEST UTILITIES *****            2007-09-06
   Test Mode ON               - Display Called Objects -          Object
                                                                          All
   Object   Library  Type         DBID   FNR S/C Ver Cat Date   Time       Calls
   *_______ DEBUG___
   DEBUG2P  DEBUG    Program        10    32 S/C 4.2 2007-08-30 13:48          1
   DEBUG2N  DEBUG    Subprogram     10    32 S/C 4.2 2007-08-30 13:48          1
   DEBUG3N  DEBUG    Subprogram     10    32 S/C 4.2 2007-08-30 13:48          1
   DEBUG4N  DEBUG    Subprogram     10    32 S/C 4.2 2007-08-30 13:48          1



   Command ===>


   Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
         Help        Exit  Last        Flip        +                      Canc
   ```

   The screen lists all objects executed: the invoking program (DEBUG2P) and all other objects invoked (DEBUG2N, DEBUG3N and DEBUG4N). It also indicates how frequently each object is invoked (`CALLS`), the type of object called, where the object is stored and under which Natural version, whether source and cataloged objects exist, and when the object was cataloged.

4. Press PF3 (Exit) or PF12 (Canc) until the NEXT prompt appears.

To check which statements are executed during program execution

1. At the NEXT prompt, enter the following:

```
TEST SET XSTAT COUNT
```

   The message `Statement execution counting started for library/object */*.` confirms that the statistics function is activated for all objects contained in the current library and all steplibs concatenated with this library.

2. Execute DEBUG2P.

   The debugger logs all statements processed by the program before the report produced by the program is output.

3. At the NEXT prompt, enter the following:

```
TEST DIS XSTAT
```

   A **List Statement Execution Statistics** screen similar to the example below appears:

```
11:39:10              ***** NATURAL TEST UTILITIES *****             2007-09-06
Test Mode ON        - List Statement Execution Statistics -     Object
                                                                            All
Co Object   Library  Type        DBID   FNR Obj.Called Exec Exec   %  Total No.
   *_______ *_______                               n Times able uted     Executions
__ DEBUG2P  DEBUG    Program       10    32          1    8    8 100          8
__ DEBUG2N  DEBUG    Subprogram    10    32          1    8    8 100          8
__ DEBUG3N  DEBUG    Subprogram    10    32          1    2    2 100          2
__ DEBUG4N  DEBUG    Subprogram    10    32          1   10    7  70          7



Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit  Last        Flip  -     +                        Canc
```

   The screen lists the number of calls (`Obj. Called n Times`), the number of executable statements (`Exec able`), the number of executed statements (`Executed`), the percentage of executed statements as related to the total number of executable statements (`%`), and the total number of executed statements (`Total No. Executions`).

4. In the **Co** column, next to `DEBUG4N`, enter the following:

```
DS
```

A statistics screen similar to the example below appears:

```
12:11:19              ***** NATURAL TEST UTILITIES *****              2007-09-06
Test Mode ON             - Display Statement Lines -             Object DEBUG4N

Line Source                                                               Count
0010 ** SUBPROGRAM 'DEBUG4N': CALCULATES SPECIAL SALARY INCREASE
0020 ***********************************************************
0030 DEFINE DATA
0040 PARAMETER
0050 1 SALARY (P7.2)
0060 END-DEFINE
0070 DECIDE FOR FIRST CONDITION                                               1
0080   WHEN SALARY < 50000                                                    1
0090     SALARY := SALARY + 1800                                   not executed
0100   WHEN SALARY < 70000                                                    1
0110     SALARY := SALARY + 1200                                   not executed
0120   WHEN SALARY < 90000                                                    1
0130     SALARY := SALARY + 600                                    not executed
0140   WHEN NONE                                                              1
0150     SALARY := SALARY + 300                                               1

Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit  Last        Flip        +                        Canc
```

The screen indicates how often a statement was executed and the executable statements that were not processed.

Additional Hints for Using the Debugger

This section provides additional hints for using the debugger.

- Time Stamps of Objects
- Saving Breakpoints and Watchpoints
- Debug Main Menu for Maintenance Functions
- Help for Commands on Maintenance Screens
- Major Functions Available during Program Interruption
- Next Option for Additional Commands During Program Interruption
- Displaying Large Variables and Arrays
- Printing Debugger Statistics
- Using the Debugger in Batch Mode

Time Stamps of Objects

A cataloged object that does not exactly correspond to the source object can cause debugging errors. If you want to guarantee that source and cataloged object correspond to each other, save and catalog them with the system command `STOW`.

For details, see the section *Operational Requirements*.

Saving Breakpoints and Watchpoints

You can save the breakpoints and watchpoints set in the current session as a debug environment and load this environment for use in a future session. This is helpful if you want to repeatedly test an application with the same debug entries.

For details, see the section *Debug Environment Maintenance*.

Debug Main Menu for Maintenance Functions

All debugger maintenance functions, such as setting a breakpoint or creating statistics, can be executed by using either a direct command or the maintenance functions provided in the **Debug Main Menu**. You open this menu by entering one of the following:

- `TEST`
  at a command prompt.
- `MENU`
  at the Command line of a debugger screen.
- `M`
  in the **Code** field of the **Debug Break** window.

Help for Commands on Maintenance Screens

For a list of direct commands available on a debugger maintenance screen, press PF1 (Help) or enter a question mark (?) in the Command line.

A debugger maintenance screen that contains list items usually also provides line commands that can be used to further process an item. You enter a line command in the **Co** column, next to the required item. For a list of valid line commands, enter a question mark (?) in this column.

Major Functions Available during Program Interruption

The major functions available during the program interruption are listed in the following section. They can be executed from either the **Debug Break** window or the Command line of a debugger maintenance screen.

Code in Debug Window	Alternative Direct Command	Function
`G`	`GO`	Continues the program execution until the next debug event occurs.
`L`	`LIST BREAK`	Lists the object source at the statement line where the debug event occurs.
`N`	`NEXT`	Executes the next break command if specified for a breakpoint or watchpoint. See also *Next Option for Additional Commands During Program Interruption*.
`R`	`RUN`	Switches test mode off and continues the program execution.
`S`	`STEP`	Processes the executable statements line by line.
`V`	`DIS VAR`	Displays a list of variables defined for the interrupted object.

Next Option for Additional Commands During Program Interruption

When displaying or modifying a breakpoint or watchpoint, you will notice that the debugger command `BREAK` is attached to each of them. This command invokes the **Debug Break** window and must not be removed. However, you can specify additional debugger commands to be executed during the program interruption after the `BREAK` command. An additional command is executed when you enter either the command `NEXT` in the Command line or the function code `N` in the **Debug Break** window.

You enter the debugger commands in the **Commands** field of the appropriate breakpoint or watchpoint maintenance screen as shown in the following example:

```
11:38:55              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON                 - Modify Breakpoint -             Object

Spy number .............   1
Initial state ........... A (A = Active, I = Inactive)
Breakpoint name ......... DEBUG1P0170_   DBID/FNR ....... 10/32
Object name ............. DEBUG1P_       Library ........ DEBUG
Line number ............. 0170
Label ................... ________________________________
Skips before execution .. ____0
Max number executions ... ____0


Commands ... BREAK___________________________________________________
             STACK___________________________________________________
             DIS VAR BONUS___________________________________________
             ________________________________________________________
             ________________________________________________________
             ________________________________________________________

Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit  Last  Save  Flip                                Canc
```

In the example above, the command `STACK` instructs the debugger to view the Natural stack. The command `DIS VAR BONUS` instructs the debugger to display the specified variable. This is helpful, for example, if you set a breakpoint in a loop and always want to view the value of one particular variable only. You then do not have to enter the `DIS VAR` command repeatedly.

For details, see the description of the field **Commands** in the sections *Fields and Columns on Breakpoint Screens* and *Fields and Columns on Watchpoint Screens*.

Displaying Large Variables and Arrays

The **Display Variable** screen shows all definitions of a variable and displays its contents in alphanumeric or hexadecimal format. For the display features available for large variables, whose contents extend beyond the current screen or variables with array definitions, see the section *Display Variable - Individual*.

Printing Debugger Statistics

You can print the statistical reports produced by the debugger or download them to a PC.

For details, see *Print Objects* in the section *Call Statistics Maintenance* and *Print Statements* in the section *Statement Execution Statistics Maintenance*.

Using the Debugger in Batch Mode

The debugger is mainly designed for interactive operations in online mode. Although you can, in principle, execute all debugger features in batch mode, processing online operations in batch (for example, the use of PF keys) can require complex batch programming. However, there are also debugger features for which batch processing is a convenient alternative. One example is collecting and printing statistical data about an application as described in *Example of Generating and Printing Statistics in Batch* in the section *Batch Processing*.

Example Sources

This section contains the source code of the example programs and subprograms required in Sessions 1 to 5.

Program DEBUG1P

```
** PROGRAM 'DEBUG1P: CALLS 'DEBUG1N' FOR SALARY AND BONUS CALCULATION
**********************************************************************
DEFINE DATA
LOCAL
1 EMPLOYEE    (A42)
1 REDEFINE EMPLOYEE
  2 NAME      (A20)
  2 NUMCHILD  (N2)
  2 ENTRYDATE (D)
  2 SALARY    (P7.2)
  2 BONUS     (P7.2)
END-DEFINE
NAME      := 'MEIER'
NUMCHILD  := 2
ENTRYDATE := D'1989-01-01'
* SALARY  := 99000
```

```
CALLNAT ’DEBUG1N’ NAME NUMCHILD ENTRYDATE SALARY BONUS
WRITE ’EMPLOYEE RECEIVES:’   SALARY
WRITE ’    PLUS BONUS OF:’   BONUS
END
```

Subprogram DEBUG1N

```
** SUBPROGRAM ’DEBUG1N’: CALCULATES BONUS AND SALARY INCREASE
************************************************************************
DEFINE DATA
PARAMETER
1 EMPLOYEE
  2 NAME      (A20)
  2 NUMCHILD  (N2)
  2 ENTRYDATE (D)
  2 SALARY    (P7.2)
  2 BONUS     (P7.2)
LOCAL
1 TARGETDATE  (D)    INIT <D’2009-01-01’>
1 DIFFERENCE  (P3.2)
1 PERCENT     (P2.2) INIT <3.5>
END-DEFINE
DIFFERENCE := (TARGETDATE - ENTRYDATE) / 365
IF DIFFERENCE GE 10        /* BONUS FOR YEARS IN COMPANY
  BONUS := SALARY * PERCENT / 100
END-IF
SALARY := SALARY + 1800    /* SALARY PLUS ANNUAL INCREASE
END
```

Program DEBUG2P

```
** PROGRAM ’DEBUG2P’: CALLS ’DEBUG2N’FOR SALARY AND BONUS CALCULATION
************************************************************************
DEFINE DATA
LOCAL
1 EMPLOYEE    (A42)
1 REDEFINE EMPLOYEE
  2 NAME      (A20)
  2 NUMCHILD  (N2)
  2 ENTRYDATE (D)
  2 SALARY    (P7.2)
  2 BONUS     (P7.2)
END-DEFINE
NAME      := ’MEIER’
NUMCHILD  := 2
ENTRYDATE := D’1989-01-01’
SALARY    := 99000
CALLNAT ’DEBUG2N’ NAME NUMCHILD ENTRYDATE SALARY BONUS
WRITE ’EMPLOYEE RECEIVES:’   SALARY
WRITE ’    PLUS BONUS OF:’   BONUS
END
```

Subprogram DEBUG2N

```
** SUBPROGRAM DEBUG2N: CALLS ’DEBUG3N’ AND ’DEBUG4N’FOR SPECIAL RATES
************************************************************************
DEFINE DATA
PARAMETER
1 EMPLOYEE
  2 NAME      (A20)
  2 NUMCHILD  (N2)
  2 ENTRYDATE (D)
  2 SALARY    (P7.2)
  2 BONUS     (P7.2)
LOCAL
1 TARGETDATE  (D)    INIT <D’2009-01-01’>
1 DIFFERENCE  (P3.2)
1 PERCENT     (P2.2) INIT <3.5>
END-DEFINE
DIFFERENCE := (TARGETDATE - ENTRYDATE) / 365
IF DIFFERENCE GE 10        /* BONUS FOR YEARS IN COMPANY
  BONUS := SALARY * PERCENT / 100
END-IF
IF NUMCHILD > 0
  CALLNAT ’DEBUG3N’ NUMCHILD BONUS     /* SPECIAL BONUS
END-IF
CALLNAT ’DEBUG4N’ SALARY               /* SPECIAL SALARY INCREASE
END
```

Subprogram DEBUG3N

```
** SUBPROGRAM ’DEBUG3N’: CALCULATES SPECIAL BONUS
************************************************************************
DEFINE DATA
PARAMETER
1 NUMCHILD (N2)
1 BONUS    (P7.2)
END-DEFINE
BONUS := BONUS + NUMCHILD * 50
END
```

Subprogram DEBUG4N

```
** SUBPROGRAM ’DEBUG4N’: CALCULATES SPECIAL SALARY INCREASE
************************************************************************
DEFINE DATA
PARAMETER
1 SALARY (P7.2)
END-DEFINE
DECIDE FOR FIRST CONDITION
  WHEN SALARY < 50000
    SALARY := SALARY + 1800
  WHEN SALARY < 70000
    SALARY := SALARY + 1200
  WHEN SALARY < 90000
    SALARY := SALARY + 600
  WHEN NONE
    SALARY := SALARY + 300
END-DECIDE
END
```