chap0101.txt Chapter 1: How to define types and data objects
chap0102.txt Chapter 1: A Few Simple Examples
chap0103.txt Chapter 1: Working with database tables and internal tables
chap0104.txt Chapter 1: Designing a report
chap0301.txt Chapter 3: The Syntax of ABAP/4 Programs
chap0401.txt Chapter 4: Three approaches to define data objects
chap0402.txt Chapter 4: Types, data, constants
chap0403.txt Chapter 4: Character types
chap0404.txt Chapter 4: Numbers
chap0405.txt Chapter 4: Date and time
chap0406.txt Chapter 4: Hexadecimal (or binary) data
chap0407.txt Chapter 4: Records and internal tables
chap0408.txt Chapter 4: Complex Non-Elementary Types and Data Objects
chap0409.txt Chapter 4: Using system fields
chap0501.txt Chapter 5: Working with tables from the Dictionary
chap0701.txt Chapter 7: Copying fields
chap0702.txt Chapter 7: Simple examples of field conversion
chap0703.txt Chapter 7: Converting character fields
chap0704.txt Chapter 7: Converting number fields
chap0705.txt Chapter 7: Converting date fields
chap0706.txt Chapter 7: Copying structured objects
chap0707.txt Chapter 7: Arithmetic Expressions and Mathematical Functions
chap0708.txt Chapter 7: String Operations
chap0709.txt Chapter 7: Special conversions
chap0801.txt Chapter 8: Using the Basic Layout Formats
chap0802.txt Chapter 8: Customizing pages
chap0803.txt Chapter 8: Skipping lines
chap0804.txt Chapter 8: Setting the layout position of fields
chap0806.txt Chapter 8: Using symbols and icons
chap0807.txt Chapter 8: Using colors
chap0808.txt Chapter 8: Type-Specific Output Options
chap0809.txt Chapter 8: Multi-Language Support
chap0901.txt Chapter 9: External flow of control (events)
chap0902.txt Chapter 9: Internal flow of control (if, case, do, while)
chap1001.txt Chapter 10: Simple form (local subroutine of a program)
chap1002.txt Chapter 10: Local data in a form
chap1003.txt Chapter 10: Using static variables
chap1004.txt Chapter 10: Using interface parameters of a form
chap1005.txt Chapter 10: Classifying parameters
chap1006.txt Chapter 10: Using table parameters
chap1007.txt Chapter 10: Type check for form parameters
chap1008.txt Chapter 10: Form parameters without type reference
chap1009.txt Chapter 10: Form parameters with generic types
chap1010.txt Chapter 10: Calling a function
chap1011.txt Chapter 10: Recursive calls
chap1101.txt Chapter 11: A simple query

chap1102.txt Chapter 11: Using an alternative work area
chap1103.txt Chapter 11: Using internal tables as snapshots of database tables
chap1104.txt Chapter 11: Using where clauses
chap1105.txt Chapter 11: Reading single entries
chap1106.txt Chapter 11: Selecting single fields
chap1107.txt Chapter 11: Getting statistical information
chap1108.txt Chapter 11: Ordering query results
chap1109.txt Chapter 11: Using Select-Options
chap1110.txt Chapter 11: Using a dynamic table name
chap1111.txt Chapter 11: Obtaining data with nested select loops
chap1112.txt Chapter 11: Using Internal Tables for Selection Criteria
chap1201.txt Chapter 12: A simple internal table
chap1202.txt Chapter 12: Internal tables with header lines
chap1203.txt Chapter 12: Filling an internal table from a database table
chap1204.txt Chapter 12: Appending single lines
chap1205.txt Chapter 12: Appending multiple lines
chap1206.txt Chapter 12: Inserting lines at a specified position
chap1207.txt Chapter 12: Inserting lines at a specified position
chap1301.txt Chapter 13: Inserting single entries in a database table
chap1302.txt Chapter 13: Inserting multiple lines in a database table
chap1303.txt Chapter 13: Updating single entries in a database table
chap1304.txt Chapter 13: Updating multiple entries in a database table
chap1305.txt Chapter 13: Modifying single entries in a database table
chap1306.txt Chapter 13: Modifying multiple entries in a database table
chap1307.txt Chapter 13: Deleting single entries from a database table
chap1308.txt Chapter 13: Deleting multiple entries from a database table
chap1401.txt Chapter 14: Exporting to the ABAP/4 Memory
chap1402.txt Chapter 14: Importing from the ABAP/4 Memory
chap1501.txt Chapter 15: Using select statements
chap1502.txt Chapter 15: Using a Logical Database
chap1503.txt Chapter 15: Using the events start-of-selection and end-of-selection
chap1504.txt Chapter 15: Working with get events
chap1601.txt Chapter 16: Parameters on the selection screen
chap1602.txt Chapter 16: Working with Select-Options
chap1603.txt Chapter 16: Selection screen events
chap1701.txt Chapter 17: Double-clicking
chap1702.txt Chapter 17: Clicking on a hotspot area
chap1703.txt Chapter 17: Pop-up Screens
chap1704.txt Chapter 17: Working with the hide command
chap1705.txt Chapter 17: Tabular lists
chap1801.txt Chapter 18: Sample report with selection criteria
chap1802.txt Chapter 18: Running a report
chap1803.txt Chapter 18: Displaying the selection screen
chap1901.txt Chapters 19-22: Sample dialog program (flight reservation)
chap2301.txt Chapter 23: Dynamic sort command
chap2302.txt Chapter 23: Dynamic sort command with several sort criteria

chap2303.txt Chapter 23: Dynamic read table command
chap2304.txt Chapter 23: Dynamic subtotals
chap2305.txt Chapter 23: Dynamic Open SQL Commands: table name
chap2306.txt Chapter 23: Dynamic Open SQL Commands: table name
chap2307.txt Chapter 23: External perform (caller)
chap2308.txt Chapter 23: External perform (called form)
chap2309.txt Chapter 23: Dynamic external perform (call back form)
chap2310.txt Chapter 23: Dynamic external perform
chap2401.txt Chapter 24: Working with Field Symbols
chap2402.txt Chapter 24: Using Field Symbols for variable parts of fields
chap2403.txt Chapter 24: Using Field Symbols for components of a structure
chap2501.txt Chapter 25: Working with temporary programs
chap2502.txt Chapter 25: Syntax errors in temporary programs
chap2503.txt Chapter 25: A real life example for using a temporary program
chap2504.txt Chapter 25: Generating a persistent program
chap2601.txt Chapter 26: Transferring data to a file
chap2602.txt Chapter 26: Reading data from a file
chap2603.txt Chapter 26: Transferring data to a file (presentation server)
chap2604.txt Chapter 26: Reading data from a file (presentation server)
chap2801.txt Chapter 28: Sample program for OLE Automation

```
*&---------------------------------------------------------------------*
*& Chapter 1: How to define types and data objects                     *
*&---------------------------------------------------------------------*
REPORT CHAP0101.

* Elementary type character, length 20
DATA CUSTOMER_NAME(25) TYPE C.

* Non-elementary type
TYPES T_NAME(25) TYPE C.
DATA NEW_CUSTOMER_NAME TYPE T_NAME.

* Reference to a data object
DATA VENDOR_NAME LIKE CUSTOMER_NAME.

* Record
DATA: BEGIN OF BOOKING,
        ID(4) TYPE C,
        FLIGHT_DATE TYPE D,
        NAME LIKE CUSTOMER_NAME,
      END OF BOOKING.

* Internal table
DATA BOOKING_TABLE LIKE BOOKING OCCURS 100.
```

```
*&---------------------------------------------------------------------*
*& Chapter 1: A Few Simple Examples                                    *
*&---------------------------------------------------------------------*
REPORT CHAP0102.

* Copying the content of one data object to another
DATA: SOURCE(10) TYPE C,
      TARGET LIKE SOURCE.
MOVE SOURCE TO TARGET.

* Displaying the contents of fields
WRITE 'ABAP/4 is easy.'.
NEW-LINE.
WRITE 'This text is displayed on a new line.'.
WRITE / 'After the symbol /, text also appears on a new line.'.

* Standard control structures (conditions and loops)
IF SOURCE = TARGET.
  WRITE / 'Fields source and target have the same content'.
ELSE.
  WRITE / 'Fields source and target do not have the same content'.
ENDIF.

DO 3 TIMES.
  WRITE / SY-INDEX.
ENDDO.

* Local subroutine of a single program
DATA: A1 TYPE I,
      A2 TYPE I.
PERFORM CALC USING    A1
            CHANGING A2.
WRITE / A2.
FORM CALC USING    F1 LIKE A1
          CHANGING F2 LIKE A2.
  F2 = F1 + ( F2 * 17 ).
ENDFORM.

* Event for drill-down facilities (reacts when a user selects a line)
AT LINE-SELECTION.
  WRITE 'This is displayed after double-clicking a line'.

*&---------------------------------------------------------------------*
*& Chapter 1: Working with database tables and internal tables         *
*&---------------------------------------------------------------------*
REPORT CHAP0103.
```

```
* Declaration of a work area for a Dictionary table
TABLES CUSTOMERS.
* Internal table used as snapshot of the database table
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100
                WITH HEADER LINE.
* Reading the entries of the database table into an internal table
SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS.
* Displaying each line of an internal table
LOOP AT ALL_CUSTOMERS.
  WRITE: / ALL_CUSTOMERS-NAME.
ENDLOOP.

*&---------------------------------------------------------------------*
*& Chapter 1: Designing a report                                       *
*&---------------------------------------------------------------------*
REPORT CHAP0104.

* Declaration of a work area for a Dictionary table
TABLES CUSTOMERS.
* Internal table used as snapshot of the database table
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100
                WITH HEADER LINE.
* Definition of input fields on the report's selection screen
SELECT-OPTIONS SNAME FOR CUSTOMERS-NAME.
* Reading the entries of the database table into an internal table
SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS
      WHERE NAME IN SNAME.
* Displaying each line of an internal table
LOOP AT ALL_CUSTOMERS.
  WRITE: / ALL_CUSTOMERS-NAME.
ENDLOOP.

*&---------------------------------------------------------------------*
*& Chapter 3: The Syntax of ABAP/4 Programs
*&---------------------------------------------------------------------*
* Declaration of the program name
REPORT CHAP0301.

* Displaying the words 'Customer list' on the screen
WRITE / 'Customer list'.

* Using an addition of the write command
WRITE AT /10 'Customer list'.

* Using single quotation marks within the text of a literal
```

```
WRITE / 'Customer''s Name'.

* Here is a comment with an asterisk in the first column
WRITE / 'Ms O''Connor'. "This is a comment at the end of the line

* A field of type character and length 40
DATA TARGET_STRING(40) TYPE C.

* Statements may extend over several lines
* (e.g., copying fields using the move command):
MOVE 'Source string'
  TO
    TARGET_STRING.
WRITE / TARGET_STRING.

* Combining Statements
WRITE: / 'Customer list',
         'Bookings'.

*&---------------------------------------------------------------------*
*& Chapter 4: Three approaches to define data objects
*&---------------------------------------------------------------------*
REPORT CHAP0401.

* 1. Elementary types
DATA: CUSTOMER_NAME_1(25) TYPE C,
      VENDOR_NAME_1(25)   TYPE C.

* 2. Reference to an existing field
DATA: CUSTOMER_NAME_2(25) TYPE C,
      VENDOR_NAME_2 LIKE CUSTOMER_NAME_2.

* 3. Reference to a non-elementary type
TYPES T_NAME(25) TYPE C.
DATA: CUSTOMER_NAME_3 TYPE T_NAME,
      VENDOR_NAME_3   TYPE T_NAME.

*&---------------------------------------------------------------------*
*& Chapter 4: Types, data, constants
*&---------------------------------------------------------------------*
REPORT CHAP0402.

* Type flag defines an abstract type
TYPES FLAG TYPE C.

* Field address_flag will allocate space in main memory at runtime
```

```
DATA ADDRESS_FLAG TYPE FLAG VALUE 'X'.

* Constants are defined like fields and cannot be changed
CONSTANTS: COMPANY_NAME(3) TYPE C VALUE 'SAP',
           MAX_COUNTER TYPE I VALUE 9999.

* Using constants to define initial values
DATA COUNTER TYPE I VALUE MAX_COUNTER.

*&---------------------------------------------------------------------*
*& Chapter 4: Character types
*&---------------------------------------------------------------------*
REPORT CHAP0403.

* Type c is the default type when no type is specified.
* Initial value is space, if it is not specified explicitly.
DATA: NAME(25) TYPE C,
      CITY(25),
      FLAG,
      SINGLE_CHARACTER VALUE 'A'.

* If the field and the initial value have different lengths, the
* initial value is either truncated or padded with blanks on the right:
DATA LANGUAGE(2) VALUE 'ABAP/4'.
WRITE / LANGUAGE.

* Maximum length 64KB
DATA MAX_CHARACTER_FIELD(65535).

* Variables of type n (numeric texts) contain strings of digits
DATA CUSTOMER_ID(8) TYPE N VALUE '87654321'.

* The default length for a field of type n is 1,
* and the default initial value is a string of zeros
DATA ZIP_CODE(5) TYPE N.
WRITE / ZIP_CODE.

* Type n fields pad the left side with zeroes
CUSTOMER_ID = '1234'.
WRITE / CUSTOMER_ID.

*&---------------------------------------------------------------------*
*& Chapter 4: Numbers
*&---------------------------------------------------------------------*
REPORT CHAP0404.
```

```
* Fields of type i (integer) are mainly used for counting
DATA: CUSTOMER_NUMBER TYPE I,
      LOOP_COUNTER TYPE I.
* Integers have a fixed length of 4 bytes.
* The initial value is zero, if it is not specified explicitly.
DATA WORD_LENGTH TYPE I VALUE 17.

* Packed numbers (type p) are a way to store numbers internally
* in a compressed form. Therefore, they cover a wide range of possible
* values can be used for all kinds of computations.
DATA NUMBER_OF_MOSQUITOES TYPE P.

* Decimal handling is supported for packed numbers
DATA AIRBAG_PRICE TYPE P DECIMALS 2 VALUE '333.22'.
WRITE / AIRBAG_PRICE.

* Default length of type p fields is 8, and the maximum length is 16,
* which can represent numbers of up to 31 digits plus the sign
DATA: PACKED_NORMAL TYPE P,
      PACKED_16(16) TYPE P.

* Floating point numbers (type f) occur in complex arithmetic
* operations. Possible values range from 1E~-307 to 1E307.
* The standard output length of fields of type f is 22.
DATA AGE_OF_EARTH TYPE F VALUE '123E+8'.
WRITE / AGE_OF_EARTH.

* The values of floating point numbers can be represented in
* different ways, but they are all equivalent:
DATA: F1 TYPE F,
      F2 TYPE F,
      F3 TYPE F.
F1 = 1.
F2 = '-12.34567'.
F3 = '-765E04'.

O/p:
F1=1.0000000000000000E+00
F2=-1.2345670000000000E+01
F3=-7.6500000000000000E+06

*&---------------------------------------------------------------------*
*& Chapter 4: Date and time
*&---------------------------------------------------------------------*
REPORT CHAP0405.
```

```
* Date fields are type d with the fixed length 8 and the internal
* representation YYYYMMDD (year, month, and day).
* The initial value of a date field is 00000000.
DATA TODAY TYPE D.

* The write command formats dates according to personal settings of
* the end user.
TODAY = SY-DATUM.
WRITE (10) TODAY.

* Using date fields to perform computations
DATA ULTIMO TYPE D.
* Set variable to first day of current month.
ULTIMO = SY-DATUM.
ULTIMO+6(2) = '01'.
* Set variable to last day of previous month.
SUBTRACT 1 FROM ULTIMO.
WRITE / ULTIMO.

* Time fields are type t with the fixed length 6
* and the format HHMMSS (hours, minutes, and seconds)
DATA MY_TIME TYPE T.
WRITE /(8) MY_TIME.

*&---------------------------------------------------------------------*
*& Chapter 4: Hexadecimal (or binary) data                             *
*&---------------------------------------------------------------------*
REPORT CHAP0406.

* Hexadecimal (or binary) data is stored in fields of type x.
* A type x field of length n contains 2n digits
* and its output length is also equal to 2n.
* For example, the bit stream 1111000010001001 can be defined as
* follows (remind that 1111 = F, 0000 = 0, 1000 = 8, 1001 = 9):
DATA XSTRING(2) TYPE X VALUE 'F089'.

*&---------------------------------------------------------------------*
*& Chapter 4: Records and internal tables
*&---------------------------------------------------------------------*
REPORT CHAP0407.

* Records (or structures) consist of a fixed number of components
DATA: BEGIN OF CUSTOMER,
      ID(8) TYPE N,
      NAME(25),
      TELEPHONE(12),
```

```
     END OF CUSTOMER.

* Working with the different components and the structure itself
DATA VENDOR LIKE CUSTOMER.
CUSTOMER-ID = '87654321'.
CUSTOMER-NAME = 'Edison'.
CUSTOMER-TELEPHONE = '111-111-1111'.
MOVE CUSTOMER TO VENDOR.
WRITE / VENDOR-NAME.

* Defining an internal table each entry having the structure of
* the record customer
DATA ALL_CUSTOMERS LIKE CUSTOMER OCCURS 100.

* Using a reference to a non-elementary type.
TYPES: BEGIN OF PERSONAL_DATA,
         NAME(25),
         CITY(25),
         STREET(30),
       END OF PERSONAL_DATA.
DATA PEOPLE TYPE PERSONAL_DATA OCCURS 300.

* Internal table with a header line, which is used as a default record
* to hold the record currently being added to the table
DATA NEW_CUSTOMERS LIKE CUSTOMER OCCURS 100
                WITH HEADER LINE.

*&---------------------------------------------------------------------*
*& Chapter 4: Complex Non-Elementary Types and Data Objects
*&---------------------------------------------------------------------*
REPORT CHAP0408.

* Nested records
TYPES: BEGIN OF ADDRESS,
         CITY(25),
         STREET(30),
       END OF ADDRESS,
       BEGIN OF PERSON,
         NAME(25),
         ADDRESS TYPE ADDRESS,
       END OF PERSON.
DATA RECEIVER TYPE PERSON.
RECEIVER-NAME = 'Smith'.
RECEIVER-ADDRESS-CITY = 'Big City'.
RECEIVER-ADDRESS-STREET = 'Main street'.
```

```
* Nested internal tables
TYPES: BEGIN OF PHONE_FAX_NUMBERS,
         COUNTRY_CODE(3) TYPE N,
         AREA_CODE(3) TYPE N,
         NUMBER(10) TYPE N,
       END OF PHONE_FAX_NUMBERS,
       BEGIN OF EMPLOYEE,
         NAME(25),
         PHONE TYPE PHONE_FAX_NUMBERS OCCURS 10,
         FAX TYPE PHONE_FAX_NUMBERS OCCURS 5,
       END OF EMPLOYEE.
DATA EMPLOYEES TYPE EMPLOYEE OCCURS 100.

*&---------------------------------------------------------------------*
*& Chapter 4: Using system fields
*&---------------------------------------------------------------------*
REPORT CHAP0409.

WRITE: / 'Current date',        SY-DATUM,
       / 'Current table index', SY-TABIX,
       / 'Loop counter',        SY-INDEX,
       / 'System return code',  SY-SUBRC.

*&---------------------------------------------------------------------*
*& Chapter 5: Working with tables from the Dictionary
*&---------------------------------------------------------------------*
REPORT CHAP0501.

* Declaration of a work area for a Dictionary table
TABLES CUSTOMERS.
* Reading all entries of the database table and displaying each entry
SELECT * FROM CUSTOMERS.
  WRITE: / CUSTOMERS-NAME.
ENDSELECT.

*&---------------------------------------------------------------------*
*& Chapter 7: Copying fields
*&---------------------------------------------------------------------*
REPORT CHAP0701.

* move fields
DATA: NAME(25),
      COUNTER TYPE I.
DATA: SOURCE LIKE NAME,
      TARGET LIKE SOURCE.
MOVE: 'Edison' TO NAME,
```

```
    17 TO COUNTER.
MOVE SOURCE TO TARGET.

* Using the compute command (keyword can be omitted)
COMPUTE TARGET = SOURCE.
TARGET = SOURCE.

* Concatenating compute commands
DATA: PHONE_1 LIKE SOURCE,
    PHONE_2 LIKE PHONE_1,
    PHONE_3 LIKE PHONE_2,
    PHONE_4 LIKE PHONE_3.
PHONE_4 = PHONE_3 = PHONE_2 = PHONE_1 = SOURCE.

*&---------------------------------------------------------------------*
*& Chapter 7: Simple examples of field conversion
*&---------------------------------------------------------------------*
REPORT CHAP0702.

* Converting to numbers during computations
DATA: NUMBER_1(4) VALUE '1771',
    NUMBER_2(3),
    RESULT TYPE I.

NUMBER_2 = '005'.
RESULT = NUMBER_1 + NUMBER_2.
WRITE / RESULT.

NUMBER_2 = '  5'.
RESULT = NUMBER_1 + NUMBER_2.
WRITE / RESULT.

* Padding character fields with blanks
DATA: OLD_CUSTOMER_NAME(10) VALUE 'Edison',
    NEW_CUSTOMER_NAME(25).
MOVE OLD_CUSTOMER_NAME TO NEW_CUSTOMER_NAME.
WRITE / NEW_CUSTOMER_NAME.

* Calculating dates
DATA: ANY_DATE TYPE D,
    SAME_DAY_OF_NEXT_WEEK TYPE D.

ANY_DATE = '19991231'.
SAME_DAY_OF_NEXT_WEEK = ANY_DATE + 7.
WRITE / SAME_DAY_OF_NEXT_WEEK.
```

```
ANY_DATE = '20000228'.
SAME_DAY_OF_NEXT_WEEK = ANY_DATE + 7.
WRITE / SAME_DAY_OF_NEXT_WEEK.


*&---------------------------------------------------------------------*
*& Chapter 7: Converting character fields
*&---------------------------------------------------------------------*
REPORT CHAP0703.

* Truncating fields or padding with blanks
DATA: SHORT_NAME(8),
      LONG_NAME(16).
MOVE 'Washington' TO: SHORT_NAME, LONG_NAME.
WRITE: / SHORT_NAME, LONG_NAME.

* Take care of intermediate steps
LONG_NAME = SHORT_NAME = 'Washington'.
WRITE: / SHORT_NAME, LONG_NAME.

*&---------------------------------------------------------------------*
*& Chapter 7: Converting number fields
*&---------------------------------------------------------------------*
REPORT CHAP0704.

* Using numeric texts and packed numbers
DATA: NO_EMPLOYEES(4)    TYPE N,
      NO_ROOMS           TYPE P,
      EMPLOYEES_PER_ROOM TYPE P DECIMALS 2.
EMPLOYEES_PER_ROOM = NO_EMPLOYEES / NO_ROOMS.

* Rounding with integers and packed numbers
DATA: INCOME             TYPE I           VALUE '10000',
      TAX                TYPE P DECIMALS 2 VALUE '0.2',
      NET_INCOME         TYPE P DECIMALS 2,
      ROUNDED_NET_INCOME TYPE I.

NET_INCOME = INCOME * ( 1 - TAX ).
ROUNDED_NET_INCOME = INCOME * ( 1 - TAX ).
WRITE: / NET_INCOME, ROUNDED_NET_INCOME.

INCOME = '10002'.
NET_INCOME = INCOME * ( 1 - TAX ).
ROUNDED_NET_INCOME = INCOME * ( 1 - TAX ).
WRITE: / NET_INCOME, ROUNDED_NET_INCOME.
----------------------------------------------------------------------*
```

```
*& Chapter 7: Converting date fields
*&---------------------------------------------------------------------*
REPORT CHAP0705.

* Using the built-in calendar
DATA: RECEIVING_DATE       TYPE D,
      LAST_ADMISSIBLE_DATE TYPE D.

RECEIVING_DATE = '19980223'.
LAST_ADMISSIBLE_DATE = RECEIVING_DATE + 10.
WRITE / LAST_ADMISSIBLE_DATE.

RECEIVING_DATE = '19980305'.
LAST_ADMISSIBLE_DATE = RECEIVING_DATE + 10.
WRITE / LAST_ADMISSIBLE_DATE.

RECEIVING_DATE = '20000223'.
LAST_ADMISSIBLE_DATE = RECEIVING_DATE + 10.
WRITE / LAST_ADMISSIBLE_DATE.


*&---------------------------------------------------------------------*
*& Chapter 7: Copying structured objects
*&---------------------------------------------------------------------*
REPORT CHAP0706.

* Using move-corresponding to copy fields with the same name
DATA: BEGIN OF MY_CUSTOMER,
        ID(8) TYPE N,
        NAME(25),
        CITY(25),
      END OF MY_CUSTOMER,
      BEGIN OF CITY_OF_CUSTOMER,
        CITY LIKE MY_CUSTOMER-CITY,
        TEXT(30),
        ID LIKE MY_CUSTOMER-ID,
      END OF CITY_OF_CUSTOMER.

MY_CUSTOMER-ID = '87654321'.
CITY_OF_CUSTOMER-TEXT = 'Old text'.
MOVE-CORRESPONDING MY_CUSTOMER TO CITY_OF_CUSTOMER.
WRITE: / 'Changed ID',     CITY_OF_CUSTOMER-ID,
       / 'Unchanged text', CITY_OF_CUSTOMER-TEXT.

* Using the move command for structures
DATA: CURRENT_CUSTOMER LIKE MY_CUSTOMER,
```

```
      BEGIN OF PREVIOUS_CUSTOMER,
        IDENTIFIER LIKE MY_CUSTOMER-ID,
        NAME       LIKE MY_CUSTOMER-NAME,
        CITY       LIKE MY_CUSTOMER-CITY,
      END OF PREVIOUS_CUSTOMER.
CURRENT_CUSTOMER-ID = '12345678'.
MOVE CURRENT_CUSTOMER TO PREVIOUS_CUSTOMER.
WRITE: / 'Changed ID', PREVIOUS_CUSTOMER-IDENTIFIER.

* Copying complete internal tables
TYPES: BEGIN OF TABLE_LINE,
         FIELD_1,
         FIELD_2 TYPE I,
       END OF TABLE_LINE.
DATA: SOURCE_TABLE TYPE TABLE_LINE OCCURS 100,
      TARGET_TABLE TYPE TABLE_LINE OCCURS 50.
MOVE SOURCE_TABLE TO TARGET_TABLE.

*&---------------------------------------------------------------------*
*& Chapter 7: Arithmetic Expressions and Mathematical Functions
*&---------------------------------------------------------------------*
REPORT CHAP0707.

* Self-explanatory formulas
DATA: A TYPE P, B LIKE A, X LIKE A, Y LIKE A,
      INCOME TYPE I, TAX TYPE P, NET_INCOME TYPE P,
      ALPHA TYPE P.
Y = A * X + B.
NET_INCOME = INCOME * ( 1 - TAX ).
Y = X * COS( ALPHA ).

* Arithmetic expressions
DATA: BLACK_SWANS TYPE I,
      WHITE_SWANS TYPE I.
DATA PERCENTAGE TYPE P DECIMALS 2.
PERCENTAGE = BLACK_SWANS * 100 / ( BLACK_SWANS + WHITE_SWANS ).

* Associative law
DATA: N1 TYPE P, N2 TYPE P, N3 TYPE P, N4 TYPE P, N5 TYPE P.
N5 = ( ( N1 - ( N2 / N3 ) ) * ( N4 + N1 ) / N5 ).
N5 =   ( N1 - N2   / N3 )   * ( N4 + N1 ) / N5.

* More formulas including mathematical functions
DATA: W TYPE P, D TYPE I, N TYPE P, Q TYPE I, R TYPE P.
Y = X * COS( ALPHA ).
A = 1.
```

```
W = EXP( B * LOG( A ) ).
D = N DIV Q.

*&---------------------------------------------------------------------*
*& Chapter 7: String Operations
*&---------------------------------------------------------------------*
REPORT CHAP0708.

* Concatenating strings without delimiter
DATA: FIRST_NAME(25), MIDDLE_NAME(2), LAST_NAME(25),
     FULL_NAME(54).
FIRST_NAME  = 'John'.
MIDDLE_NAME = 'F.'.
LAST_NAME   = 'Kennedy'.
CONCATENATE FIRST_NAME MIDDLE_NAME LAST_NAME INTO
FULL_NAME.
WRITE / FULL_NAME.

* Concatenating strings with delimiter
DATA: DIRECTORY_1(2), DIRECTORY_2(10), FILE_NAME(10),
     PATH(24).
DIRECTORY_1 = 'a:'.
DIRECTORY_2 = 'usr'.
FILE_NAME = 'programs'.
CONCATENATE DIRECTORY_1 DIRECTORY_2 FILE_NAME
          INTO PATH
          SEPARATED BY '\'.
WRITE / PATH.

* Splitting strings
DATA: LIST(40),
     NAME_1(25), NAME_2(25), NAME_3(25).
LIST = 'Edison,Smith,Young'.
SPLIT LIST AT ',' INTO NAME_1 NAME_2 NAME_3.
WRITE: / NAME_1, NAME_2, NAME_3.

* Splitting strings with result in an internal table
DATA NAMES LIKE NAME_1 OCCURS 10 WITH HEADER LINE.
LIST = 'Edison,Smith,Young,Edwards'.
SPLIT LIST AT ',' INTO TABLE NAMES.
LOOP AT NAMES.
  WRITE / NAMES.
ENDLOOP.

* Shifting strings by a fixed number of places
NAME_1 = 'Edison'.
```

```
NAME_2 = 'Smith'.
NAME_3 = 'Young'.
SHIFT NAME_1.
SHIFT NAME_2 BY 3 PLACES.
SHIFT NAME_3 RIGHT.
WRITE: / NAME_1, NAME_2, NAME_3.

* Shifting strings up to a substring
NAMES = 'Alexander Bill Charles'.
SHIFT NAMES UP TO 'Bill'.
WRITE / NAMES.

* Shifting strings deleting blanks
NAMES = 'Joanne___'.
SHIFT NAMES RIGHT DELETING TRAILING SPACE.
WRITE / NAMES.

* Replacing and translating characters in strings
DATA: STRING(80),
      EXPRESSION(30).
STRING = 'Variable: &. The variable & is substituted later.'.
REPLACE '&' WITH 'X' INTO STRING.
WRITE / STRING.
TRANSLATE STRING USING '&X'.
WRITE / STRING.
EXPRESSION = 'a ** 2 + b ** 2 = c ** 2'.
TRANSLATE EXPRESSION USING 'axbycz'.
WRITE / EXPRESSION.

* Searching for strings in fields or internal tables
DATA TEXT(100) VALUE 'Texas California New Mexico Louisiana Oregon'.
SEARCH TEXT FOR 'California'.
IF SY-SUBRC NE 0. WRITE 'Not found'. ENDIF.
SEARCH TEXT FOR 'cAliforniA'.
IF SY-SUBRC NE 0. WRITE 'Not found'. ENDIF.
SEARCH TEXT FOR 'New M'.
IF SY-SUBRC NE 0. WRITE 'Not found'. ENDIF.

* Working with parts of fields
DATA: S(8) VALUE 'ABCDEFGH',
      T(8) VALUE '12345678',
      OFF1 TYPE I, OFF2 TYPE I,
      LEN1 TYPE I, LEN2 TYPE I.

OFF1 = 2.
LEN1 = 3.
```

```
OFF2 = 4.
LEN2 = 3.
MOVE S+OFF1(LEN1) TO T+OFF2(LEN2).
WRITE / T.

*&---------------------------------------------------------------------*
*& Chapter 7: Special conversions
*&---------------------------------------------------------------------*
REPORT CHAP0709.

* Converting type c to type n
DATA: SCN(4) VALUE '12x4',
      T1CN(2) TYPE N,
      T2CN(6) TYPE N.
MOVE: SCN TO T1CN,
      SCN TO T2CN.

* Converting type n to type c
DATA: SNC(4) TYPE N VALUE '0124',
      T1NC(2),
      T2NC(6).
MOVE: SNC TO T1NC,
      SNC TO T2NC.

* Converting type n to type p
DATA: SNP(6) TYPE N VALUE '012345',
      T1NP(10) TYPE P,
      T2NP(2) TYPE P.
MOVE  SNP TO T1NP.
*move snp to t2np. "This produces a runtime error when activated!

* Converting type p to type n
DATA: SPN(4) TYPE P VALUE 124,
      T1PN(2) TYPE N,
      T2PN(6) TYPE N.
MOVE: SPN TO T1PN,
      SPN TO T2PN.

WRITE 'Program finished'.

*&---------------------------------------------------------------------*
*& Chapter 8: Using the Basic Layout Formats
*&---------------------------------------------------------------------*
REPORT CHAP0801.

* Simple output containing the current date
```

```
WRITE: 'This is the current date:', SY-DATUM.

* Displaying fields according to their type
DATA: STRING(20),
      INT TYPE I,
      PACKED_NUMBER TYPE P DECIMALS 2,
      DATE LIKE SY-DATUM.
STRING = 'Customer list'.
INT = 17.
PACKED_NUMBER = 5 / 4.
DATE = SY-DATUM + 30.
WRITE: / STRING, INT, PACKED_NUMBER, DATE.

*&---------------------------------------------------------------------*
*& Chapter 8: Customizing pages
*&---------------------------------------------------------------------*

* suppress the standard header of a page
REPORT CHAP0802 NO STANDARD PAGE HEADING..

* Define a counter for the output page
DATA COUNTER(9) TYPE N.

* Start a new page and set the line size
NEW-PAGE LINE-SIZE 44.

* Display the counter 40 times
DO 40 TIMES.
  COUNTER = SY-INDEX.
  WRITE COUNTER.
ENDDO.

*&---------------------------------------------------------------------*
*& Chapter 8: Skipping lines
*&---------------------------------------------------------------------*
REPORT CHAP0803.

WRITE 'This string will appear on the first line.'.
NEW-LINE.
WRITE: 'New line',
     / 'Yet another line'.
SKIP 3.
WRITE / 'skip 3 produces three empty lines.'.


*&---------------------------------------------------------------------*
```

```
*& Chapter 8: Setting the layout position of fields
*&---------------------------------------------------------------------*
REPORT CHAP0804.

DATA: POS TYPE I,
      LEN TYPE I.

WRITE AT 5 'position 5'.
POS = 20.
WRITE AT POS 'position 20'.
POSITION 40.
WRITE 'position 40'.

POS = 10. LEN = 20.
NEW-LINE.
WRITE AT POS(LEN) 'position 10, length 20'.

*&---------------------------------------------------------------------*
*& Chapter 8: Using symbols and icons
*&---------------------------------------------------------------------*
REPORT CHAP0806.

* Declaring symbols and icons
INCLUDE: <SYMBOL>, <ICON>.

* Displaying symbols and icons
WRITE: / SYM_PHONE     AS SYMBOL, 'telephone',
       / SYM_FAX       AS SYMBOL, 'fax machine',
       / SYM_LEFT_HAND AS SYMBOL, 'hand pointing left',
       / SYM_CAUTION   AS SYMBOL, 'caution',
       / ICON_CHECKED  AS ICON,   'checked; okay',
       / ICON_DELETE   AS ICON,   'delete',
       / ICON_PRINT    AS ICON,   'print'.

*&---------------------------------------------------------------------*
*& Chapter 8: Using colors
*&---------------------------------------------------------------------*
REPORT CHAP0807.

* Display the header using an appropriate color (grayish blue)
WRITE 'Header' COLOR COL_HEADING.

* Switch the standard color
FORMAT COLOR COL_TOTAL.

* Make the color less bright
```

```
WRITE / 'total sum' COLOR COL_TOTAL INTENSIFIED OFF.

* Using different colors
FORMAT COLOR COL_HEADING.
WRITE / 'Header'.
FORMAT COLOR OFF.
SKIP.
WRITE: / 'Key field' COLOR COL_KEY,
         'Background',
         'Negative' COLOR COL_NEGATIVE,
       / 'Total sum' COLOR COL_TOTAL INTENSIFIED OFF.

*&---------------------------------------------------------------------*
*& Chapter 8: Type-Specific Output Options
*&---------------------------------------------------------------------*
REPORT CHAP0808.

* Specifying a format template
DATA TIME TYPE T VALUE '154633'.
WRITE AT (8) TIME USING EDIT MASK '__:__:__'.

* Using decimals
DATA PACKED_NUMBER TYPE P VALUE 123.
WRITE PACKED_NUMBER DECIMALS 2.

*&---------------------------------------------------------------------*
*& Chapter 8: Multi-Language Support
*&---------------------------------------------------------------------*
REPORT CHAP0809.

* Using text symbols
WRITE: / 'Literal without text symbol',
       / 'Original text of the source code'(001),
       / TEXT-002.

*&---------------------------------------------------------------------*
*& Chapter 9: External flow of control (events)
*&---------------------------------------------------------------------*
REPORT CHAP0901.

* Display a list of customers
TABLES CUSTOMERS.
SELECT * FROM CUSTOMERS.
  WRITE / CUSTOMERS-NAME.
ENDSELECT.
```

```
* Event for drill down
AT LINE-SELECTION.
  WRITE: / 'This line appears after drill-down'.

*&---------------------------------------------------------------------*
*& Chapter 9: Internal flow of control (if, case, do, while)
*&---------------------------------------------------------------------*
REPORT CHAP0902.

* Declarations for later use
TABLES CUSTOMERS.
DATA: COLOR(10) VALUE 'yellow',
      N(4) TYPE N VALUE '123',
      P TYPE P,
      C4(4) VALUE '124',
      C5(5) VALUE '00124',
      SQUARE_NUMBER TYPE I,
      X TYPE I,
      Y TYPE I.

* Using a condition (e.g., business class or not)
IF CUSTOMERS-CUSTTYPE = 'B'.
* book business class
  WRITE 'B'.
ELSE.
* book economy class
  WRITE 'Something else'.
ENDIF.

* Nested if clauses
IF N > 0.
  N = N + 1.
ELSE.
  IF N = 0.
    WRITE / 'zero'.
  ELSE.
    N = N - 1.
  ENDIF.
ENDIF.

* Using elseif instead of a nested if clauses
IF N > 0.
  N = N + 1.
ELSEIF N = 0.
  WRITE / 'zero'.
ELSE.
```

```
  N = N - 1.
ENDIF.

* Using a case clause
CASE COLOR.
  WHEN 'red'.    WRITE 'color is red'.
  WHEN 'green'.  WRITE 'color is green'.
  WHEN 'yellow'. WRITE 'color is yellow'.
  WHEN OTHERS.   WRITE 'non-standard color'.
ENDCASE.

* Some logical expressions in if clauses
IF N IS INITIAL.
  WRITE 'initial'.
ELSEIF N LT 0
    OR N GT 5.
  WRITE / 'less than zero or greater than 5'.
ELSE.
  WRITE / 'something else'.
ENDIF.

IF N > P.
  WRITE / 'n is greater than p'.
ENDIF.

* Conversion in an expression
IF C4 = C5.
  WRITE / 'c4 and c5 are equal'.
ENDIF.

* Comparing character strings
DATA: A(6) VALUE 'ABAP/4',
      RESULT(6).
IF A CA 'XP'.
  RESULT = A+SY-FDPOS(2).
  WRITE / RESULT.
ENDIF.

IF A CO 'ABP'.
  WRITE / 'a only contains A,B, and P'.
ENDIF.

IF A CS 'BAP'.
  WRITE / 'a contains the string BAP'.
ENDIF.
```

```
IF A CP '*AP++'.
  WRITE / 'a contains AP followed by two more characters'.
ENDIF.

* Unconditional loop
DO 100 TIMES.
  SQUARE_NUMBER = SY-INDEX ** 2.
  WRITE / SQUARE_NUMBER.
ENDDO.

* Terminating a loop
DO.
* terminate loop after 5 steps or when the color is red
  IF SY-INDEX > 5 OR COLOR = 'red'. EXIT. ENDIF.
* main loop step
  WRITE / SY-INDEX.
ENDDO.

* Using a conditional loop.
X = Y - 2.
WHILE X <> Y.
  X = Y + 1.
  WRITE / X.
  IF X > Y. EXIT. ENDIF.
ENDWHILE.

*&---------------------------------------------------------------------*
*& Chapter 10: Simple form (local subroutine of a program)
*&---------------------------------------------------------------------*
REPORT CHAP1001.

* Global field of the program
DATA FLAG VALUE 'G'.
* Displaying the global field
WRITE FLAG.
* Calling a form
PERFORM SET_FLAG.
* Displaying the global field again
WRITE FLAG.

* Defining a form
FORM SET_FLAG.
* Changing and displaying the global field
  FLAG = 'L'.
  WRITE FLAG.
ENDFORM.
```

```
*&---------------------------------------------------------------------*
*& Chapter 10: Local data in a form
*&---------------------------------------------------------------------*
REPORT CHAP1002.

* Global field of the program
DATA FLAG VALUE 'G'.
* Displaying the global field
WRITE FLAG.
* Calling a form
PERFORM WRITE_FLAG.
* Displaying the global field again
WRITE FLAG.

* Defining a form with local data
FORM WRITE_FLAG.
* Local data
  DATA L_FLAG.
* Changing and displaying local data
  L_FLAG = 'L'.
  WRITE L_FLAG.
ENDFORM.

*&---------------------------------------------------------------------*
*& Chapter 10: Using static variables
*&---------------------------------------------------------------------*
REPORT CHAP1003.
* Calling a form twice
PERFORM COUNT.
PERFORM COUNT.
* Defining a form with a static variable
FORM COUNT.
  STATICS CALLS TYPE I.
  CALLS = CALLS + 1.
  WRITE CALLS.
ENDFORM.

*&---------------------------------------------------------------------*
*& Chapter 10: Using interface parameters of a form
*&---------------------------------------------------------------------*
REPORT CHAP1004.

* Types and data for later use
TYPES: T_NAME(20).
DATA: NAME_1 TYPE T_NAME VALUE 'A',
```

```
     NAME_2 TYPE T_NAME VALUE 'B'.
* Calling a form with different parameters
PERFORM SET_NAME CHANGING NAME_1.
PERFORM SET_NAME CHANGING NAME_2.
* Defining a form with a parameter
FORM SET_NAME
    CHANGING F_NAME TYPE T_NAME.
  WRITE F_NAME.
  F_NAME = 'Smith'.
  WRITE F_NAME.
ENDFORM.

*&---------------------------------------------------------------------*
*& Chapter 10: Classifying parameters
*&---------------------------------------------------------------------*
REPORT CHAP1005.

* Data declarations for later use
DATA: A1 TYPE P VALUE 2,
      A2 TYPE P VALUE 4,
      A3 TYPE P VALUE 8.
* Calling a form with different parameter types
PERFORM CALC USING    A1
                      A2
             CHANGING A3.
* Displaying the result
WRITE A3.
* Defining a form with different parameter types
FORM CALC USING    VALUE(F1) LIKE A1
                   F2  LIKE A2
          CHANGING VALUE(F3) LIKE A3.
  F3 = F1 + ( F2 * F3 ).
ENDFORM.

*&---------------------------------------------------------------------*
*& Chapter 10: Using table parameters
*&---------------------------------------------------------------------*
REPORT CHAP1006.

* Work area of database table and internal table for later use
TABLES CUSTOMERS.
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 50 WITH HEADER LINE.
* Calling a form with a table parameter
PERFORM READ_CUSTOMERS TABLES ALL_CUSTOMERS.
LOOP AT ALL_CUSTOMERS.
  WRITE / ALL_CUSTOMERS-NAME.
```

```
ENDLOOP.
* Defining a form with a table parameter
FORM READ_CUSTOMERS TABLES F_CUSTOMERS STRUCTURE
ALL_CUSTOMERS.
  SELECT * FROM CUSTOMERS INTO TABLE F_CUSTOMERS.
ENDFORM.
*&---------------------------------------------------------------------*
*& Chapter 10: Type check for form parameters
*&---------------------------------------------------------------------*
REPORT CHAP1007.
* Types and variables for later use
TYPES: T_NAME_1(20),
       T_NAME_2(20).
DATA: NAME_1 TYPE T_NAME_1,
      NAME_2 TYPE T_NAME_2.
* Calling forms with different actual parameters
PERFORM SET_NAME_LIKE CHANGING NAME_1.
PERFORM SET_NAME_LIKE CHANGING NAME_2.
PERFORM SET_NAME_TYPE CHANGING NAME_1.
PERFORM SET_NAME_TYPE CHANGING NAME_2.
* Form definition with type reference via like
FORM SET_NAME_LIKE CHANGING F_NAME LIKE NAME_2.
  F_NAME = 'Smith'.
ENDFORM.
* Form definition with type reference via type
FORM SET_NAME_TYPE CHANGING F_NAME TYPE T_NAME_2.
  F_NAME = 'Smith'.
ENDFORM.

*&---------------------------------------------------------------------*
*& Chapter 10: Form parameters without type reference
*&---------------------------------------------------------------------*
REPORT CHAP1008.
* Variable for later use
DATA: STRING_1(2) VALUE 'AB',
      STRING_2(8) VALUE '  ABAP/4'.
* Calling forms with different actual parameters
PERFORM WRITE_FIRST_CHARACTER CHANGING: STRING_1,
                                       STRING_2.
* Form parameters without type reference
FORM WRITE_FIRST_CHARACTER CHANGING F_STRING.
  SHIFT F_STRING LEFT DELETING LEADING SPACE.
  WRITE AT (1) F_STRING.
ENDFORM.

*&---------------------------------------------------------------------*
```

```
*& Chapter 10: Form parameters with generic types
*&---------------------------------------------------------------------*
REPORT CHAP1009.
* Variable for later use
DATA: SHORT_STRING(3) VALUE 'AB',
      SHORT_NUMBER(3) TYPE N VALUE '0',
      ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100.
* Calling forms with different actual parameters
* Correct call (actual paramter is of type c)
PERFORM WRITE_FIRST_CHARACTER CHANGING SHORT_STRING.
* Inccorrect call (actual paramter is not of type c)
*perform write_first_character changing short_number.
* Correct call (actual paramter is a table)
PERFORM SORT_AND_SEARCH_IN_TABLE
        CHANGING ALL_CUSTOMERS.

* Form parameters with generic types
FORM WRITE_FIRST_CHARACTER CHANGING F_STRING TYPE C.
  SHIFT F_STRING LEFT DELETING LEADING SPACE.
  WRITE AT (1) F_STRING.
ENDFORM.

FORM SORT_AND_SEARCH_IN_TABLE
    CHANGING F_TABLE TYPE TABLE.
  SORT F_TABLE.
  SEARCH F_TABLE FOR 'Smith'.
ENDFORM.

*&---------------------------------------------------------------------*
*& Chapter 10: Calling a function
*&---------------------------------------------------------------------*
REPORT CHAP1010.
* Variable for later use
DATA: X TYPE I VALUE 5,
      Y LIKE X VALUE 7,
      MAXIMUM LIKE X.
* Calling a function
CALL FUNCTION 'MAX_TEST'
    EXPORTING
        A       = X
        B       = Y
    IMPORTING
        MAX     = MAXIMUM.
WRITE MAXIMUM.

*&---------------------------------------------------------------------*
```

*& **Chapter 10: Recursive calls**
&---
```
REPORT CHAP1011.
* Variable for later use
DATA: NUMBER TYPE I VALUE 5,
      RESULT TYPE I VALUE 1.
* Calling a form from the main program
PERFORM FACTORIAL.
WRITE RESULT.
* Defining a form with a recursive call
FORM FACTORIAL.
  IF NUMBER <= 1.
    EXIT.
  ENDIF.
  RESULT = RESULT * NUMBER.
  NUMBER = NUMBER - 1.
  PERFORM FACTORIAL.
ENDFORM.
```

&---
*& **Chapter 11: A simple query**
&---
```
REPORT CHAP1101.
* Work area for a database table
TABLES CUSTOMERS.
* Reading all entries of the database table
SELECT * FROM CUSTOMERS.
  WRITE: / CUSTOMERS-NAME.
ENDSELECT.
```

&---
*& **Chapter 11: Using an alternative work area**
&---
```
REPORT CHAP1102.
* Work area for a database table
TABLES CUSTOMERS.
* alternative work area
DATA MY_CUSTOMER LIKE CUSTOMERS.
* Reading all entries of the database table
SELECT * FROM CUSTOMERS INTO MY_CUSTOMER.
  WRITE: / MY_CUSTOMER-NAME.
ENDSELECT.
```

&---

*& **Chapter 11: Using internal tables as snapshots of database tables**
&---
```
REPORT CHAP1103.
* Work area for a database table
TABLES CUSTOMERS.
* Internal table
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100
               WITH HEADER LINE.
* Filling the internal table with all entries of the database table
SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS.
* Displaying the contents of the internal table
LOOP AT ALL_CUSTOMERS.
  WRITE: / ALL_CUSTOMERS-NAME.
ENDLOOP.
```

&---
*& **Chapter 11: Using where clauses**
&---
```
REPORT CHAP1104.
* Work areas
TABLES: BOOKINGS, CUSTOMERS.
* Internal tables
DATA CUSTOMER_ORDERS LIKE BOOKINGS OCCURS 100
                WITH HEADER LINE.
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100
               WITH HEADER LINE.

* Selecting data with a simple where clause
SELECT * FROM BOOKINGS INTO TABLE CUSTOMER_ORDERS
      WHERE ORDER_DATE = '19990101'.
* Displaying the result
LOOP AT CUSTOMER_ORDERS.
  WRITE / CUSTOMER_ORDERS-FLDATE.
ENDLOOP.

* Selecting data with a complex where clause
SELECT * FROM BOOKINGS INTO TABLE CUSTOMER_ORDERS
        WHERE CUSTOMID   =  '87654321'
          AND ORDER_DATE >= '19990101'.
* Displaying the result
SKIP.
LOOP AT CUSTOMER_ORDERS.
  WRITE / CUSTOMER_ORDERS-FLDATE.
ENDLOOP.

* Selecting data with a complex where clause
```

```
SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS
      WHERE NAME LIKE 'E%'.
* Displaying the result
SKIP.
LOOP AT ALL_CUSTOMERS.
  WRITE / ALL_CUSTOMERS-NAME.
ENDLOOP.

*&---------------------------------------------------------------------*
*& Chapter 11: Reading single entries
*&---------------------------------------------------------------------*
REPORT CHAP1105.
* Work area for a database table
TABLES CUSTOMERS.
* Reading a single entry
SELECT SINGLE * FROM CUSTOMERS
      WHERE ID = '87654321'.
IF SY-SUBRC = 0.
  WRITE CUSTOMERS-NAME.
ELSE.
  WRITE 'Customer not found.'.
ENDIF.

*&---------------------------------------------------------------------*
*& Chapter 11: Selecting single fields
*&---------------------------------------------------------------------*
REPORT CHAP1106.
* Work area for a database table
TABLES CUSTOMERS.
* Selecting single fields
DATA: CID   LIKE CUSTOMERS-ID,
     CNAME LIKE CUSTOMERS-NAME.
SELECT ID NAME INTO (CID,CNAME) FROM CUSTOMERS.
  WRITE: / CID, CNAME.
ENDSELECT.

*&---------------------------------------------------------------------*
*& Chapter 11: Getting statistical information
*&---------------------------------------------------------------------*
REPORT CHAP1107.
* Work area for a database table
TABLES: BOOKINGS, ACTFLI.
* Variables for later use
DATA: COUNT_BOOKINGS TYPE I,
     AVERAGE_SEATS_OCCUPIED LIKE ACTFLI-SEATSOCC,
     MAX_SEATS            LIKE ACTFLI-SEATSMAX.
```

```
* Getting the number of selected entries
SELECT COUNT(*) FROM BOOKINGS INTO COUNT_BOOKINGS
      WHERE ORDER_DATE >= '19990101'.
WRITE COUNT_BOOKINGS.
* Average and maximum
SELECT AVG( SEATSOCC ) MAX( SEATSMAX ) FROM ACTFLI
      INTO (AVERAGE_SEATS_OCCUPIED,MAX_SEATS).
WRITE: / AVERAGE_SEATS_OCCUPIED, MAX_SEATS.

*&---------------------------------------------------------------------*
*& Chapter 11: Ordering query results
*&---------------------------------------------------------------------*
REPORT CHAP1108.
* Work area
TABLES CUSTOMERS.
* Reading table entries in a specified order
SELECT * FROM CUSTOMERS
      ORDER BY CITY NAME.
  WRITE: / CUSTOMERS-CITY,
           CUSTOMERS-NAME.
ENDSELECT.

*&---------------------------------------------------------------------*
*& Chapter 11: Using Select-Options
*&---------------------------------------------------------------------*
REPORT CHAP1109.
* Work area
TABLES CUSTOMERS.
* Specifiying a Select-Option
SELECT-OPTIONS SNAME FOR CUSTOMERS-NAME.
* Internal table for later use
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100
                WITH HEADER LINE.
* Reading table entries according to a Select-Option
SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS
      WHERE NAME IN SNAME.
* Displaying the reuslt
LOOP AT ALL_CUSTOMERS.
  WRITE: / ALL_CUSTOMERS-CITY,
           ALL_CUSTOMERS-NAME.
ENDLOOP.



*&---------------------------------------------------------------------*
```

```
*& Chapter 11: Using a dynamic table name
*&---------------------------------------------------------------------*
REPORT CHAP1110.
* Variables for later use
DATA: TABLENAME(10),
      COUNT_ROWS TYPE I.
* Setting the table name dynamically
MOVE 'CUSTOMERS' TO TABLENAME.
* Selecting data
SELECT COUNT( * ) FROM (TABLENAME) INTO COUNT_ROWS.
WRITE: TABLENAME, COUNT_ROWS.

*&---------------------------------------------------------------------*
*& Chapter 11: Obtaining data with nested select loops
*&---------------------------------------------------------------------*
REPORT CHAP1111.
* Work areas
TABLES: CUSTOMERS, BOOKINGS.
* Reading entries from both database table
SELECT * FROM CUSTOMERS.
  SELECT * FROM BOOKINGS
         WHERE CUSTOMID   = CUSTOMERS-ID
           AND ORDER_DATE = '19990101'.
    WRITE: / CUSTOMERS-NAME,
             BOOKINGS-FLDATE.
  ENDSELECT.
ENDSELECT.

*&---------------------------------------------------------------------*
*& Chapter 11: Using Internal Tables for Selection Criteria
*&---------------------------------------------------------------------*
REPORT CHAP1112.
* Work areas
TABLES: CUSTOMERS, BOOKINGS.
* Internal tables
DATA: ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100
                    WITH HEADER LINE,
      ALL_BOOKINGS  LIKE BOOKINGS OCCURS 500
                    WITH HEADER LINE.
* Reading entries from both database table
SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS.
SELECT * FROM BOOKINGS  INTO TABLE ALL_BOOKINGS
       FOR ALL ENTRIES IN ALL_CUSTOMERS
       WHERE CUSTOMID   = ALL_CUSTOMERS-ID
         AND ORDER_DATE = '19990101'.
* Displaying the result
```

```
LOOP AT ALL_CUSTOMERS.
  LOOP AT ALL_BOOKINGS
      WHERE CUSTOMID = ALL_CUSTOMERS-ID.
    WRITE: / ALL_CUSTOMERS-NAME,
           ALL_BOOKINGS-FLDATE.
  ENDLOOP.
ENDLOOP.

*&---------------------------------------------------------------------*
*& Chapter 12: A simple internal table
*&---------------------------------------------------------------------*
REPORT CHAP1201.
* Work area for a database table
TABLES CUSTOMERS.
* Defining an internal table
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100.
* Reading all entries of the database table into the internal table
SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS.

*&---------------------------------------------------------------------*
*& Chapter 12: Internal tables with header lines
*&---------------------------------------------------------------------*
REPORT CHAP1202.
* Work area for a database table
TABLES CUSTOMERS.
* Defining an internal table with header line
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100
                WITH HEADER LINE.
* Reading all entries of the database table into the internal table
SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS.

*&---------------------------------------------------------------------*
*& Chapter 12: Filling an internal table from a database table
*&---------------------------------------------------------------------*
REPORT CHAP1203.
* Work area for a database table
TABLES CUSTOMERS.
* Defining an internal table with header line
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100
                WITH HEADER LINE.
* Filling the internal table (previous content overwritten)
SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS.
* Filling the internal table (previous content kept)
SELECT * FROM CUSTOMERS APPENDING TABLE ALL_CUSTOMERS.
* Displaying the result
LOOP AT ALL_CUSTOMERS.
```

```
  WRITE / ALL_CUSTOMERS-NAME.
ENDLOOP.

*&---------------------------------------------------------------------*
*& Chapter 12: Appending single lines
*&---------------------------------------------------------------------*
REPORT CHAP1204.
* Work area for a database table
TABLES CUSTOMERS.
* Types for later use
TYPES: BEGIN OF T_CUSTOMER_CITY,
         ID   LIKE CUSTOMERS-ID,
         CITY LIKE CUSTOMERS-CITY,
       END OF T_CUSTOMER_CITY.
* Internal table with two columns
DATA CUSTOMER_CITIES TYPE T_CUSTOMER_CITY OCCURS 100
                     WITH HEADER LINE.
* Filling the internal table
SELECT * FROM CUSTOMERS.
  MOVE-CORRESPONDING CUSTOMERS TO CUSTOMER_CITIES.
  APPEND CUSTOMER_CITIES.
ENDSELECT.
* Displaying the result
LOOP AT CUSTOMER_CITIES.
  WRITE / CUSTOMER_CITIES-CITY.
ENDLOOP.

*&---------------------------------------------------------------------*
*& Chapter 12: Appending multiple lines
*&---------------------------------------------------------------------*
REPORT CHAP1205.
* Work area for a database table
TABLES CUSTOMERS.
* Defining internal tables
DATA: ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100
                    WITH HEADER LINE,
      OLD_CUSTOMERS LIKE CUSTOMERS OCCURS 10
                    WITH HEADER LINE.
* Filling both internal tables
SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS.
SELECT * FROM CUSTOMERS INTO TABLE OLD_CUSTOMERS.
* Appending one internal table to the other
APPEND LINES OF OLD_CUSTOMERS TO ALL_CUSTOMERS.
* Displaying the result
LOOP AT ALL_CUSTOMERS.
  WRITE / ALL_CUSTOMERS-NAME.
```

```
ENDLOOP.

*&---------------------------------------------------------------------*
*& Chapter 12: Inserting lines at a specified position
*&---------------------------------------------------------------------*
REPORT CHAP1206.
* Work area for a database table
TABLES CUSTOMERS.
* Types for later use
TYPES: BEGIN OF T_CUSTOMER_CITY,
         ID   LIKE CUSTOMERS-ID,
         CITY LIKE CUSTOMERS-CITY,
       END OF T_CUSTOMER_CITY.
* Internal table with two columns
DATA CUSTOMER_CITIES TYPE T_CUSTOMER_CITY OCCURS 100
                     WITH HEADER LINE.
* Filling the internal table
SELECT * FROM CUSTOMERS.
  MOVE-CORRESPONDING CUSTOMERS TO CUSTOMER_CITIES.
  APPEND CUSTOMER_CITIES.
ENDSELECT.
* Inserting a line at a specified position
CUSTOMER_CITIES-ID   = '00000005'.
CUSTOMER_CITIES-CITY = 'Pleasant Site'.
INSERT CUSTOMER_CITIES INDEX 3.
* Displaying the result
LOOP AT CUSTOMER_CITIES.
  WRITE / CUSTOMER_CITIES-CITY.
ENDLOOP.

*&---------------------------------------------------------------------*
*& Chapter 12: Inserting lines at a specified position
*&---------------------------------------------------------------------*
REPORT CHAP1206.
* Work area for a database table
TABLES CUSTOMERS.
* Types for later use
TYPES: BEGIN OF T_CUSTOMER_CITY,
         ID   LIKE CUSTOMERS-ID,
         CITY LIKE CUSTOMERS-CITY,
       END OF T_CUSTOMER_CITY.
* Internal table with two columns
DATA CUSTOMER_CITIES TYPE T_CUSTOMER_CITY OCCURS 100
                     WITH HEADER LINE.
* Filling the internal table
SELECT * FROM CUSTOMERS.
```

```
  MOVE-CORRESPONDING CUSTOMERS TO CUSTOMER_CITIES.
  APPEND CUSTOMER_CITIES.
ENDSELECT.
* Inserting a line at a specified position
CUSTOMER_CITIES-ID   = '00000005'.
CUSTOMER_CITIES-CITY = 'Pleasant Site'.
INSERT CUSTOMER_CITIES INDEX 3.
* Displaying the result
LOOP AT CUSTOMER_CITIES.
  WRITE / CUSTOMER_CITIES-CITY.
ENDLOOP.

*&---------------------------------------------------------------------*
*& Chapter 13: Inserting single entries in a database table
*&---------------------------------------------------------------------*
REPORT CHAP1301.
* Work area
TABLES CUSTOMERS.
* Record used as alternative work area
DATA MY_CUSTOMER LIKE CUSTOMERS.
* Inserting one entry from the work area
CUSTOMERS-ID = '12345678'.
CUSTOMERS-NAME = 'Brown'.
INSERT CUSTOMERS.
IF SY-SUBRC <> 0.
  WRITE: / 'Entry already exists:', CUSTOMERS-ID.
ENDIF.
* Inserting one entry from the record
MY_CUSTOMER-ID = '11111111'.
MY_CUSTOMER-NAME = 'Green'.
INSERT INTO CUSTOMERS VALUES MY_CUSTOMER.
IF SY-SUBRC <> 0.
  WRITE: / 'Entry already exists:', MY_CUSTOMER-ID.
ENDIF.

*&---------------------------------------------------------------------*
*& Chapter 13: Inserting multiple entries in a database table
*&---------------------------------------------------------------------*
REPORT CHAP1302.
* Work area
TABLES CUSTOMERS.
* Internal table for new entries
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100
                WITH HEADER LINE.
* Filling the internal table
ALL_CUSTOMERS-ID   = '12345678'.
```

```
ALL_CUSTOMERS-NAME = 'Brown'.
APPEND ALL_CUSTOMERS.
ALL_CUSTOMERS-ID   = '11111111'.
ALL_CUSTOMERS-NAME = 'Green'.
APPEND ALL_CUSTOMERS.
ALL_CUSTOMERS-ID   = '12121212'.
ALL_CUSTOMERS-NAME = 'White'.
APPEND ALL_CUSTOMERS.
* Inserting the internal table
INSERT CUSTOMERS FROM TABLE ALL_CUSTOMERS.

*&---------------------------------------------------------------------*
*& Chapter 13: Updating single entries in a database table
*&---------------------------------------------------------------------*
REPORT CHAP1303.
* Work area
TABLES CUSTOMERS.
* Record used as alternative work area
DATA MY_CUSTOMER LIKE CUSTOMERS.
* Updating one entry from the work area
CUSTOMERS-ID = '12345678'.
CUSTOMERS-CITY = 'Village'.
UPDATE CUSTOMERS.
IF SY-SUBRC <> 0.
  WRITE:  'Entry not existing:', CUSTOMERS-ID.
ENDIF.

*&---------------------------------------------------------------------*
*& Chapter 13: Updating multiple entries in a database table
*&---------------------------------------------------------------------*
REPORT CHAP1304.
* Work area
TABLES CUSTOMERS.
* Internal table for changed entries
DATA CHANGED_CUSTOMERS LIKE CUSTOMERS OCCURS 50
                  WITH HEADER LINE.
* Filling the internal table
SELECT * FROM CUSTOMERS INTO TABLE CHANGED_CUSTOMERS
      WHERE CITY = SPACE.
LOOP AT CHANGED_CUSTOMERS.
  CHANGED_CUSTOMERS-CITY = 'City unknown'.
  MODIFY CHANGED_CUSTOMERS.
ENDLOOP.
* Updating the database table with values from the internal table
UPDATE CUSTOMERS FROM TABLE CHANGED_CUSTOMERS.
* Updating the database table according to a where condition
```

```
UPDATE CUSTOMERS SET   CITY = 'City unknown'
              WHERE CITY = SPACE.

*&---------------------------------------------------------------------*
*& Chapter 13: Modifying single entries in a database table
*&---------------------------------------------------------------------*
REPORT CHAP1305.
* Work area
TABLES CUSTOMERS.
* Modifying an entry
CUSTOMERS-ID   = '12345678'.
CUSTOMERS-CITY = 'Village'.
MODIFY CUSTOMERS.

*&---------------------------------------------------------------------*
*& Chapter 13: Modifying multiple entries in a database table
*&---------------------------------------------------------------------*
REPORT CHAP1306.
* Work area
TABLES CUSTOMERS.
* Internal table for changed entries
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 50
               WITH HEADER LINE.
* Filling the internal table
SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS
      WHERE CITY = SPACE.
ALL_CUSTOMERS-ID   = '04295119'.
ALL_CUSTOMERS-NAME = 'Gray'.
APPEND ALL_CUSTOMERS.
LOOP AT ALL_CUSTOMERS.
  ALL_CUSTOMERS-CITY = 'City unknown'.
  MODIFY ALL_CUSTOMERS.
ENDLOOP.
* Modifying the database table with values from the internal table
MODIFY CUSTOMERS FROM TABLE ALL_CUSTOMERS.

*&---------------------------------------------------------------------*
*& Chapter 13: Deleting single entries from a database table
*&---------------------------------------------------------------------*
REPORT CHAP1307.
* Work area
TABLES CUSTOMERS.
* Deleting an entry
CUSTOMERS-ID = '12345678'.
DELETE CUSTOMERS.
```

```
*&---------------------------------------------------------------------*
*& Chapter 13: Deleting multiple entries from a database table
*&---------------------------------------------------------------------*
REPORT CHAP1308.
* Work area
TABLES CUSTOMERS.
* Internal table for deleted entries
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 50
                WITH HEADER LINE.
* Filling the internal table
SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS
      WHERE CITY = SPACE.
* Deleting entries with values from the internal table
DELETE CUSTOMERS FROM TABLE ALL_CUSTOMERS.
* Deleting entries according to a where condition
DELETE FROM CUSTOMERS
      WHERE ID LIKE '1%'.

*&---------------------------------------------------------------------*
*& Chapter 14: Exporting to the ABAP/4 Memory
*&---------------------------------------------------------------------*
REPORT CHAP1401.
* Work areas
TABLES: CUSTOMERS, BOOKINGS.
* Internal tables which will be exported
DATA: ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100
                WITH HEADER LINE,
     ALL_BOOKINGS  LIKE BOOKINGS  OCCURS 10
                WITH HEADER LINE.
* Filling the internal tables
SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS.
SELECT * FROM BOOKINGS  INTO TABLE ALL_BOOKINGS.
* Exporting to the ABAP/4 Memory
EXPORT ALL_CUSTOMERS ALL_BOOKINGS
      TO MEMORY ID 'CUSTBOOK'.
* Displaying the result
LOOP AT ALL_CUSTOMERS.
  WRITE / ALL_CUSTOMERS-NAME.
ENDLOOP.
LOOP AT ALL_BOOKINGS.
  WRITE / ALL_BOOKINGS-FLDATE.
ENDLOOP.



*&---------------------------------------------------------------------*
```

*& **Chapter 14: Importing from the ABAP/4 Memory**
&---
```
REPORT CHAP1402.
* Internal tables which will be imported
DATA: ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100
                WITH HEADER LINE,
     ALL_BOOKINGS  LIKE BOOKINGS  OCCURS 10
                WITH HEADER LINE,
     NEW_BOOKINGS LIKE BOOKINGS OCCURS 50
                WITH HEADER LINE.
* Importing from the ABAP/4 Memory
IMPORT ALL_CUSTOMERS ALL_BOOKINGS
      FROM MEMORY ID 'CUSTBOOK'.
IF SY-SUBRC NE 0.
  WRITE 'Import failed.'.
ENDIF.
* Skipping and renaming objects at import
IMPORT ALL_BOOKINGS TO NEW_BOOKINGS
      FROM MEMORY ID 'CUSTBOOK'.
* Displaying the result
LOOP AT ALL_CUSTOMERS.
  WRITE / ALL_CUSTOMERS-NAME.
ENDLOOP.
LOOP AT ALL_BOOKINGS.
  WRITE / ALL_BOOKINGS-FLDATE.
ENDLOOP.
LOOP AT NEW_BOOKINGS.
  WRITE / NEW_BOOKINGS-FLDATE.
ENDLOOP.
```

&---
*& **Chapter 15: Using select statements**
&---
```
REPORT CHAP1501.
* Work areas
TABLES: CUSTOMERS, BOOKINGS.
* Reading data
SELECT * FROM CUSTOMERS.
  WRITE / CUSTOMERS-NAME.
  SELECT * FROM BOOKINGS
        WHERE CUSTOMID = CUSTOMERS-ID
        AND   FLDATE     > '19990501'
        AND   ORDER_DATE = '19990101'.
    WRITE: AT /3 BOOKINGS-FLDATE.
  ENDSELECT.
ENDSELECT.
```

```
*&---------------------------------------------------------------------*
*& Chapter 15: Using a Logical Database
*&---------------------------------------------------------------------*
REPORT CHAP1502.
* Work areas
TABLES: CUSTOMERS, BOOKINGS.
* Reading data
GET CUSTOMERS.
  WRITE / CUSTOMERS-NAME.
GET BOOKINGS.
  WRITE: AT /3 BOOKINGS-FLDATE.

*&---------------------------------------------------------------------*
*& Chapter 15: Using the events start-of-selection and end-of-selection
*&---------------------------------------------------------------------*
REPORT CHAP1503.
* Work area
TABLES BOOKINGS.
* Initial processing
START-OF-SELECTION.
  WRITE / 'Start'.
* Reading data
GET BOOKINGS.
  WRITE: AT /3 BOOKINGS-FLDATE.
* Final processing
END-OF-SELECTION.
  WRITE / 'Finished'.

*&---------------------------------------------------------------------*
*& Chapter 15: Working with get events
*&---------------------------------------------------------------------*
REPORT CHAP1504.
* Work areas
TABLES BOOKINGS.
* Reading data
GET BOOKINGS.
  WRITE / BOOKINGS-FLDATE.

*&---------------------------------------------------------------------*
*& Chapter 16: Parameters on the selection screen
*&---------------------------------------------------------------------*
REPORT CHAP1601.
* Defining parameters
PARAMETERS:
  P_CITY LIKE CUSTOMERS-CITY,
```

```
  P_FLDATE LIKE ACTFLI-FLDATE DEFAULT '19991231',
  P_FLAG AS CHECKBOX DEFAULT 'X'.
* Using Parameters
IF P_CITY = 'Big City'.
 WRITE 'Input value of Parameter p_city is Big City'.
ENDIF.


*&---------------------------------------------------------------------*
*& Chapter 16: Working with Select-Options
*&---------------------------------------------------------------------*
REPORT CHAP1602.
* Work area
TABLES CUSTOMERS.
* Defining Select-Options
SELECT-OPTIONS S_NAME FOR CUSTOMERS-NAME.
SELECT * FROM CUSTOMERS
      WHERE NAME IN S_NAME.
  WRITE / CUSTOMERS-NAME.
ENDSELECT.

*&---------------------------------------------------------------------*
*& Chapter 16: Selection screen events
*&---------------------------------------------------------------------*
REPORT CHAP1603 MESSAGE-ID SU.
* Worka area
TABLES CUSTOMERS.
* Selection criteria
PARAMETERS PCODE LIKE CUSTOMERS-POSTCODE.
SELECT-OPTIONS S_NAME FOR CUSTOMERS-NAME.
* Variables
DATA: PARAMETER_LENGTH TYPE I,
     NUMBER_OF_SELECTIONS TYPE I.
* Checking user input in Parameter pcode
AT SELECTION-SCREEN ON PCODE.
  PARAMETER_LENGTH = STRLEN( PCODE ).
  IF PARAMETER_LENGTH < 5.
    MESSAGE E000 WITH 'ZIP code invalid'.
  ENDIF.
* Checking user input in Select-Option s_name
AT SELECTION-SCREEN ON S_NAME.
  DESCRIBE TABLE S_NAME LINES NUMBER_OF_SELECTIONS.
  IF NUMBER_OF_SELECTIONS = 0.
    MESSAGE E000 WITH 'Please specify name of customer'.
  ENDIF.
* Processing data
```

```
START-OF-SELECTION.
  SELECT * FROM CUSTOMERS
        WHERE NAME IN S_NAME.
    WRITE / CUSTOMERS-NAME.
  ENDSELECT.

*&---------------------------------------------------------------------*
*& Chapter 17: Double-clicking
*&---------------------------------------------------------------------*
REPORT CHAP1701.
START-OF-SELECTION.
  WRITE 'Basic list'.

AT LINE-SELECTION.
  WRITE 'New list after double-click'.

*&---------------------------------------------------------------------*
*& Chapter 17: Clicking on a hotspot area
*&---------------------------------------------------------------------*
REPORT CHAP1702.
* work area
TABLES CUSTOMERS.
* Processing data
START-OF-SELECTION.
  SELECT * FROM CUSTOMERS.
    WRITE / CUSTOMERS-NAME HOTSPOT ON.
  ENDSELECT.
* Single click
AT LINE-SELECTION.
  WRITE 'New list after single-click on a hotspot area'.

*&---------------------------------------------------------------------*
*& Chapter 17: Pop-up Screens
*&---------------------------------------------------------------------*
REPORT CHAP1703.
* work area
TABLES CUSTOMERS.
* Processing data
START-OF-SELECTION.
  SELECT * FROM CUSTOMERS.
    WRITE / CUSTOMERS-NAME HOTSPOT ON.
  ENDSELECT.
* Single click
AT LINE-SELECTION.
  WINDOW STARTING AT 10 10
         ENDING   AT 40 20.
```

```
  WRITE 'This is my first window'.

*&---------------------------------------------------------------------*
*& Chapter 17: Working with the hide command
*&---------------------------------------------------------------------*
REPORT CHAP1704.
* work area
TABLES CUSTOMERS.
* Internal table
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100
                WITH HEADER LINE.
* Processing data
START-OF-SELECTION.
  SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS.
  LOOP AT ALL_CUSTOMERS.
    WRITE / ALL_CUSTOMERS-NAME HOTSPOT ON.
    HIDE ALL_CUSTOMERS-ID.
  ENDLOOP.
* Detail information
AT LINE-SELECTION.
  WRITE: / 'Customer detail information:',
         ALL_CUSTOMERS-NAME,
         ALL_CUSTOMERS-CITY,
         ALL_CUSTOMERS-TELEPHONE.

*&---------------------------------------------------------------------*
*& Chapter 17: Tabular lists
*&---------------------------------------------------------------------*
REPORT CHAP1706 NO STANDARD PAGE HEADING.
CONSTANTS MY_LINE_SIZE TYPE I VALUE 40.
DATA SQUARE TYPE I.
NEW-PAGE LINE-SIZE MY_LINE_SIZE.
ULINE.
FORMAT COLOR COL_HEADING.
WRITE: / SY-VLINE,
         'Numbers and their squares',
         AT MY_LINE_SIZE SY-VLINE.
FORMAT COLOR OFF.
ULINE.
DO 20 TIMES.
  SQUARE = SY-INDEX ** 2.
  WRITE: / SY-VLINE,
           SY-INDEX COLOR COL_KEY,
           SY-VLINE,
           SQUARE,
         AT MY_LINE_SIZE SY-VLINE.
```

```
ENDDO.
ULINE.

*&---------------------------------------------------------------------*
*& Chapter 18: Sample report with selection criteria
*&---------------------------------------------------------------------*
REPORT CHAP1801.
TABLES: CUSTOMERS, BOOKINGS.
PARAMETERS P_DATE TYPE D.
SELECT-OPTIONS S_NAME FOR CUSTOMERS-NAME.
SELECT * FROM CUSTOMERS
      WHERE NAME IN S_NAME.
  WRITE / CUSTOMERS-NAME.
  SELECT * FROM BOOKINGS
        WHERE ORDER_DATE = P_DATE
        AND   CUSTOMID   = CUSTOMERS-ID.
    WRITE: / BOOKINGS-CARRID,
             BOOKINGS-CONNID,
             BOOKINGS-FLDATE.
  ENDSELECT.
ENDSELECT.

*&---------------------------------------------------------------------*
*& Chapter 18: Running a report
*&---------------------------------------------------------------------*
REPORT CHAP1802.
* Work area
TABLES CUSTOMERS.
* Selection criteria
SELECT-OPTIONS S_NAME FOR CUSTOMERS-NAME.
* Running report chap1801 with specified selection criteria
SUBMIT CHAP1801
      WITH P_DATE = SY-DATUM
      WITH S_NAME IN S_NAME.

*&---------------------------------------------------------------------*
*& Chapter 18: Displaying the selection screen
*&---------------------------------------------------------------------*
REPORT CHAP1803.
* Work area
TABLES CUSTOMERS.
* Selection criteria
SELECT-OPTIONS S_NAME FOR CUSTOMERS-NAME.
* running chap1801 and displaying the selection screen
SUBMIT CHAP1801
      VIA SELECTION-SCREEN
```

```
      WITH P_DATE = SY-DATUM
      WITH S_NAME IN S_NAME.

*&---------------------------------------------------------------------*
*& Chapters 19-22: Sample dialog program (flight reservation)
*&---------------------------------------------------------------------*
* This program source contains all modules and subroutines of the
* flight reservation program, but screen and GUI status definitions
* are not included.
*&---------------------------------------------------------------------*

*&---------------------------------------------------------------------*
*& Include MSABBTOP                          Modulpool        SAPMSABB *
*&                                                                    *
*&---------------------------------------------------------------------*
PROGRAM SAPMSABB MESSAGE-ID SU.

* Database tables
TABLES:
  PLANFLI,                  " Flight connections
  ACTFLI,                   " Dates of flights
  CUSTOMERS,                " Customer data
  BOOKINGS.                 " Booking informations

* User defined types
TYPES:
  BOOLEAN         TYPE C,         " true = 'X', false otherwise
  SCREEN_CODE     LIKE SY-DYNNR,
  STATUS_CODE     LIKE SY-PFKEY,
  TITLEBAR_CODE(3) TYPE N,
  FUNCTION_CODE(4) TYPE C.

DATA:
* Global data
  FCODE          TYPE FUNCTION_CODE,
  FLAG_FOUND     TYPE BOOLEAN,
  FLAG_KEEP_INPUT TYPE BOOLEAN,

* Screen fields which do not come from the dictionary
  SEATS_FREE     TYPE I,
  SEATS_REQUIRED TYPE I.

DATA:
* table of all customers
  ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100
                WITH HEADER LINE.
```

```
CONSTANTS:
* Boolean constants
  TRUE            TYPE C VALUE 'X',
  FALSE           TYPE C VALUE ' ',

* Screen codes
  C_SCREEN_REQUEST TYPE SCREEN_CODE VALUE '0100',
  C_SCREEN_BOOKING TYPE SCREEN_CODE VALUE '0200',

* Status codes
  C_STATUS_REQUEST TYPE STATUS_CODE VALUE 'REQUEST',
  C_STATUS_BOOKING TYPE STATUS_CODE VALUE 'BOOKING',
  C_STATUS_LIST    TYPE STATUS_CODE VALUE 'LIST',

* Titlebar codes
  C_TITLE_REQUEST        TYPE TITLEBAR_CODE VALUE '100',
  C_TITLE_BOOKING        TYPE TITLEBAR_CODE VALUE '200',
  C_TITLE_LIST_CUSTOMERS TYPE TITLEBAR_CODE VALUE '110',

* Function codes
  C_FCODE_REQUEST  TYPE FUNCTION_CODE VALUE 'REQU',
  C_FCODE_BOOKING  TYPE FUNCTION_CODE VALUE 'BOOK',
  C_FCODE_CLEAR    TYPE FUNCTION_CODE VALUE 'CLEA',
  C_FCODE_BACK     TYPE FUNCTION_CODE VALUE 'BACK',
  C_FCODE_EXIT     TYPE FUNCTION_CODE VALUE 'EXIT'.

*---------------------------------------------------------------------
***INCLUDE MSABBI01 .
*---------------------------------------------------------------------

*&---------------------------------------------------------------------*
*&      Module  EXIT_SCREEN  INPUT
*&---------------------------------------------------------------------*
*       Handles back, exit, and clear command.                        *
*----------------------------------------------------------------------*
MODULE EXIT_SCREEN INPUT.
  CASE FCODE.
    WHEN C_FCODE_Clear.
*     Just clear the input fields
      CASE SY-DYNNR.
        WHEN c_screen_request.  CLEAR: PLANFLI, ACTFLI.
        WHEN C_SCREEN_BOOKING.  CLEAR CUSTOMERS.
                                SEATS_REQUIRED = 1.
      ENDCASE.
    WHEN C_FCODE_BACK.
```

```
*     Switch to previous screen.
      CASE SY-DYNNR.
        WHEN c_screen_request.  SET SCREEN 0.
        WHEN C_SCREEN_BOOKING.  FLAG_KEEP_INPUT = FALSE.
                          SET SCREEN c_screen_request.
      ENDCASE.
    WHEN C_FCODE_EXIT.
*     Exit program
      SET SCREEN 0.
  ENDCASE.
  LEAVE SCREEN.
ENDMODULE.                " EXIT_SCREEN  INPUT

*&---------------------------------------------------------------------*
*&      Module  ACTION_REQUEST  INPUT
*&---------------------------------------------------------------------*
*       Check input data to represent a valid flight connection.      *
*----------------------------------------------------------------------*
MODULE ACTION_REQUEST INPUT.
  CASE FCODE.
    WHEN C_FCODE_REQUEST.
*     Check for a valid connection
      PERFORM FLIGHT_REQUEST CHANGING FLAG_FOUND.

      IF FLAG_FOUND = FALSE.
        MESSAGE E001.         " Flight connection not available
      ELSE.
        SET SCREEN 200.       " Next screen: entering customer data
        LEAVE SCREEN.
      ENDIF.
  ENDCASE.
ENDMODULE.                " ACTION_REQUEST  INPUT

*&---------------------------------------------------------------------*
*&      Module  ACTION_BOOKING  INPUT
*&---------------------------------------------------------------------*
*       Get customer data to make the booking.                        *
*----------------------------------------------------------------------*
MODULE ACTION_BOOKING INPUT.
  CASE FCODE.
    WHEN C_FCODE_BOOKING.
*     Make the booking
      PERFORM FLIGHT_BOOKING.

*     Return to previous screen for another request.
      FLAG_KEEP_INPUT = FALSE.
```

```
    SET SCREEN C_SCREEN_REQUEST.
    LEAVE SCREEN.
  ENDCASE.
ENDMODULE.                 " ACTION_BOOKING  INPUT

*&---------------------------------------------------------------------*
*&      Module  VALUES_CUSTOMERS  INPUT
*&---------------------------------------------------------------------*
*       Display list of customers                                     *
*----------------------------------------------------------------------*
MODULE VALUES_CUSTOMERS INPUT.
* Display all customers in a dialog box
  CALL SCREEN 110
       STARTING AT 10 10
       ENDING   AT 30 20.
ENDMODULE.                 " VALUES_CUSTOMERS  INPUT

*&---------------------------------------------------------------------*
*&      Module  ACTION_LIST  INPUT
*&---------------------------------------------------------------------*
*       Write list of customers                                       *
*----------------------------------------------------------------------*
MODULE ACTION_LIST INPUT.

* Read all customers
  SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS.

* Write all customers
  NEW-PAGE NO-TITLE.
  LOOP AT ALL_CUSTOMERS.
    WRITE / ALL_CUSTOMERS-NAME.
    HIDE ALL_CUSTOMERS-ID.
  ENDLOOP.

ENDMODULE.                 " ACTION_LIST  INPUT

*----------------------------------------------------------------------
***INCLUDE MSABBO01 .
*----------------------------------------------------------------------

*&---------------------------------------------------------------------*
*&      Module  INIT_REQUEST  OUTPUT
*&---------------------------------------------------------------------*
*       Initializes titlebar and status of the screen for a flight    *
*       request.                                                      *
*----------------------------------------------------------------------*
```

```
MODULE INIT_REQUEST OUTPUT.
* Initialize titlebar and status
  SET TITLEBAR  C_TITLE_REQUEST.
  SET PF-STATUS C_STATUS_REQUEST.
ENDMODULE.                " INIT_REQUEST  OUTPUT

*&---------------------------------------------------------------------*
*&      Module  INIT_BOOKING  OUTPUT
*&---------------------------------------------------------------------*
*       Initializes titlebar and status of the screen for a flight     *
*       request. The fields for the number of free and required seats  *
*       and for the customer's data are initialized.                   *
*----------------------------------------------------------------------*
MODULE INIT_BOOKING OUTPUT.
* Initialize titlebar and status
  SET TITLEBAR  C_TITLE_BOOKING.
  SET PF-STATUS C_STATUS_BOOKING.

* Get number of seats free
  SEATS_FREE = ACTFLI-SEATSMAX - ACTFLI-SEATSOCC.

* While processing the booking screen, the contents of the
* input fields are kept, e.g. if ENTER is pressed.
  IF FLAG_KEEP_INPUT = FALSE.
    CLEAR CUSTOMERS.
    SEATS_REQUIRED = 1.
    FLAG_KEEP_INPUT = TRUE.  " kept while processing the screen
  ENDIF.
ENDMODULE.                " INIT_BOOKING  OUTPUT
*&---------------------------------------------------------------------*
*&      Module  INIT_LIST  OUTPUT
*&---------------------------------------------------------------------*
*       Write list of customers                                        *
*----------------------------------------------------------------------*
MODULE INIT_LIST OUTPUT.

* Initialize titlebar and status
  SET TITLEBAR  C_TITLE_LIST_CUSTOMERS.
  SET PF-STATUS C_STATUS_LIST.

* Prepare list
  SUPPRESS DIALOG.
  LEAVE TO LIST-PROCESSING AND RETURN TO SCREEN 0.

ENDMODULE.                " INIT_LIST  OUTPUT
```

```
*---------------------------------------------------------------------
***INCLUDE MSABBF01 .
*---------------------------------------------------------------------

*&---------------------------------------------------------------------*
*&      Form  FLIGHT_REQUEST
*&---------------------------------------------------------------------*
*       Check for a valid connection                                  *
*----------------------------------------------------------------------*
FORM FLIGHT_REQUEST CHANGING F_FLAG_FOUND TYPE BOOLEAN.

* First check for a valid connection, i.e. check if carrier CARRID
* offers a flight starting in city CITYFROM with destination CITYTO.
  F_FLAG_FOUND = FALSE.
  SELECT       * FROM  PLANFLI
         WHERE  CARRID     = PLANFLI-CARRID
         AND    CITYFROM   = PLANFLI-CITYFROM
         AND    CITYTO     = PLANFLI-CITYTO.

*   Then check if the connection is also offered for the
*   specified date.
    SELECT SINGLE * FROM  ACTFLI
           WHERE  CARRID     = PLANFLI-CARRID
           AND    CONNID     = PLANFLI-CONNID
           AND    FLDATE     = ACTFLI-FLDATE.
    IF SY-SUBRC = 0.
      F_FLAG_FOUND = TRUE.
      EXIT.
    ENDIF.
  ENDSELECT.
ENDFORM.                              " FLIGHT_REQUEST


*&---------------------------------------------------------------------*
*&      Form  FLIGHT_BOOKING
*&---------------------------------------------------------------------*
*       Insert the booking into database tables                       *
*----------------------------------------------------------------------*
FORM FLIGHT_BOOKING.

  DATA: L_CUSTOMERS  LIKE CUSTOMERS OCCURS 10
                            WITH HEADER LINE,
        L_LINE_COUNT TYPE I,
        L_INDEX      LIKE SY-TABIX.

* 1. Check if seats are available
```

```
  IF SEATS_REQUIRED > SEATS_FREE.
    MESSAGE E003 WITH SEATS_FREE.       " Not enough seats
  endif.

* 2. Booking is only allowed for registered customers. Thus,
*    check if a customer id is available in table CUSTOMERS.
  SELECT * FROM  CUSTOMERS INTO TABLE L_CUSTOMERS
         WHERE NAME = CUSTOMERS-NAME.

* Check number of matching entries
  DESCRIBE TABLE L_CUSTOMERS LINES L_LINE_COUNT.
  IF L_LINE_COUNT = 0.
    message e004 with customers-name.  " Customer id not available
  ELSEIF L_LINE_COUNT > 1.
*   Process dialog to select the appropriate customer
    ...      " To be implemented: sets L_INDEX
  endif.

  L_INDEX = 1.     " Must be deleted if L_INDEX is set above
  READ TABLE L_CUSTOMERS INDEX L_INDEX.
  IF SY-SUBRC <> 0.
    MESSAGE A006.     " Internal booking error: missing entry
  ENDIF.

* 3. Update bookings information in table BOOKINGS
  MOVE-CORRESPONDING ACTFLI TO BOOKINGS.
  PERFORM SET_BOOKID CHANGING BOOKINGS-BOOKID.
  BOOKINGS-CUSTOMID   = L_CUSTOMERS-ID.
  BOOKINGS-ORDER_DATE = SY-DATUM.

  INSERT BOOKINGS.    " Optional: call function in update task
  IF SY-SUBRC <> 0.
    MESSAGE A005.     " Internal booking error: duplicate entries
  ENDIF.

* 4. Update number of occupied seats in table ACTFLI
  ADD SEATS_REQUIRED TO ACTFLI-SEATSOCC.
  UPDATE ACTFLI.      " Optional: call function in update task
  IF SY-SUBRC <> 0.
    MESSAGE A006.     " Internal booking error: missing entry
  ENDIF.

* Optional: using an update task
* call function 'ABAP_BOOK_INSERT_BOOKINGS'
*      in update task
*      exporting
```

```
*          i_bookings = bookings
*          i_actfli   = actfli.
* commit work.

* 5. Message: booking successful for customer ...
  MESSAGE I002 WITH ACTFLI-CONNID CUSTOMERS-NAME.



ENDFORM.                    " FLIGHT_BOOKING

*&---------------------------------------------------------------------*
*&      Form  SET_BOOKID
*&---------------------------------------------------------------------*
*       Determine a new booking id.                                   *
*----------------------------------------------------------------------*
FORM SET_BOOKID CHANGING F_BOOKID LIKE BOOKINGS-BOOKID.

* Get maximum bookid and increment it by 1
  SELECT MAX( BOOKID ) INTO (F_BOOKID) FROM BOOKINGS.
  ADD 1 TO F_BOOKID.
ENDFORM.                    " SET_BOOKID

*---------------------------------------------------------------------
***INCLUDE MSABBE01 .
*---------------------------------------------------------------------

*&---------------------------------------------------------------------*
*&   Event AT LINE-SELECTION
*&---------------------------------------------------------------------*

at line-selection.
* Get selected customer and display name
  read table all_customers with key id = all_customers-id.
  if sy-subrc = 0.
    customers-name = all_customers-name.
  endif.
  leave to screen 0.

*&---------------------------------------------------------------------*
*& Chapter 23: Dynamic sort command
*&---------------------------------------------------------------------*
REPORT CHAP2301.
* Parameter for the sort criterion, can be modified by the end user
PARAMETERS COLUMN(10) DEFAULT 'NAME'.
* Declarations for later use
```

```
TABLES CUSTOMERS.
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100
                WITH HEADER LINE.
* Filling the internal table
SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS.
* Dynamic sort
SORT ALL_CUSTOMERS BY (COLUMN).
* Displaying the result
LOOP AT ALL_CUSTOMERS.
  WRITE: / ALL_CUSTOMERS-ID,
           ALL_CUSTOMERS-NAME,
           ALL_CUSTOMERS-CITY,
           ALL_CUSTOMERS-TELEPHONE.
ENDLOOP.
*&---------------------------------------------------------------------*
```
***& Chapter 23: Dynamic sort command with several sort criteria**
```
*&---------------------------------------------------------------------*
REPORT CHAP2302.
* Parameters for the sort criterion, can be modified by the end user
PARAMETERS: COLUMN1(10)         DEFAULT 'NAME',
            COLUMN2 LIKE COLUMN1 DEFAULT 'ID'.
* Declarations for later use
TABLES CUSTOMERS.
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100
                WITH HEADER LINE.
* Filling the internal table
SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS.
* Dynamic sort with two sort criteria
SORT ALL_CUSTOMERS BY (COLUMN1) (COLUMN2) DESCENDING.
* Displaying the result
LOOP AT ALL_CUSTOMERS.
  WRITE: / ALL_CUSTOMERS-ID,
           ALL_CUSTOMERS-NAME,
           ALL_CUSTOMERS-CITY,
           ALL_CUSTOMERS-TELEPHONE.
ENDLOOP.


*&---------------------------------------------------------------------*
*& Chapter 23: Dynamic read table command
*&---------------------------------------------------------------------*
REPORT CHAP2303.
* Parameters for reading a single line, can be modified by the end user
PARAMETERS: KEY1(10)        DEFAULT 'NAME',
            VALUE1(25),
            KEY2   LIKE KEY1 DEFAULT 'ID',
```

```
          VALUE2 LIKE VALUE1.
* Declarations for later use
TABLES CUSTOMERS.
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100
               WITH HEADER LINE.
* Filling the internal table
SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS.
* Dynamic read table command
READ TABLE ALL_CUSTOMERS
  WITH KEY (KEY1) = VALUE1
         (KEY2) = VALUE2.
* Displaying the result
IF SY-SUBRC EQ 0.
  WRITE: / ALL_CUSTOMERS-ID,
         ALL_CUSTOMERS-NAME,
         ALL_CUSTOMERS-CITY,
         ALL_CUSTOMERS-TELEPHONE.
ELSE.
  WRITE 'Entry not found'.
ENDIF.

*&---------------------------------------------------------------------*
*& Chapter 23: Dynamic subtotals
*&---------------------------------------------------------------------*
REPORT CHAP2304.
TABLES ACTFLI.
DATA MY_FLIGHTS LIKE ACTFLI OCCURS 10
            WITH HEADER LINE.
DATA SUM_OCCUPIED_SEATS LIKE MY_FLIGHTS-SEATSOCC.
DATA COLUMN(30).
SELECT * FROM ACTFLI INTO TABLE MY_FLIGHTS
      ORDER BY PRIMARY KEY.
LOOP AT MY_FLIGHTS.
  WRITE: / MY_FLIGHTS-CARRID,
         MY_FLIGHTS-CONNID,
         MY_FLIGHTS-FLDATE.
ENDLOOP.
AT LINE-SELECTION.
* Display subtotals according to end user's selection
GET CURSOR FIELD COLUMN.
SHIFT COLUMN UP TO '-'.
SHIFT COLUMN.
LOOP AT MY_FLIGHTS.
  AT NEW (COLUMN).
    NEW-PAGE.
    WRITE / MY_FLIGHTS-CARRID.
```

```
   CLEAR SUM_OCCUPIED_SEATS.
  ENDAT.
  ADD MY_FLIGHTS-SEATSOCC TO SUM_OCCUPIED_SEATS.
  WRITE / MY_FLIGHTS-SEATSOCC.
  AT END OF (COLUMN).
    WRITE: / 'Occupied seats total:', SUM_OCCUPIED_SEATS.
  ENDAT.
ENDLOOP.

*&---------------------------------------------------------------------*
*& Chapter 23: Dynamic Open SQL Commands: table name
*&---------------------------------------------------------------------*
REPORT CHAP2305.

PARAMETERS TABLENAME(10) DEFAULT 'CUSTOMERS'.
DATA COUNT_ROWS TYPE I.
SELECT COUNT( * ) FROM (TABLENAME) INTO COUNT_ROWS.
WRITE: TABLENAME, COUNT_ROWS.

DATA WHERE_TAB(80) OCCURS 10 WITH HEADER LINE.
APPEND 'name like "E%"' TO WHERE_TAB.
APPEND 'and city like "S%"' TO WHERE_TAB.
TABLES CUSTOMERS.
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100.
SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS
      WHERE ID BETWEEN 1 AND 999
      AND (WHERE_TAB).

*&---------------------------------------------------------------------*
*& Chapter 23: Dynamic Open SQL Commands: table name
*&---------------------------------------------------------------------*
REPORT CHAP2305.

PARAMETERS TABLENAME(10) DEFAULT 'CUSTOMERS'.
DATA COUNT_ROWS TYPE I.
SELECT COUNT( * ) FROM (TABLENAME) INTO COUNT_ROWS.
WRITE: TABLENAME, COUNT_ROWS.

DATA WHERE_TAB(80) OCCURS 10 WITH HEADER LINE.
APPEND 'name like "E%"' TO WHERE_TAB.
APPEND 'and city like "S%"' TO WHERE_TAB.
TABLES CUSTOMERS.
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100.
SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS
      WHERE ID BETWEEN 1 AND 999
      AND (WHERE_TAB).
```

```
*&---------------------------------------------------------------------*
*& Chapter 23: External perform (caller)
*&---------------------------------------------------------------------*
REPORT CHAP2307.
* List of the current program
WRITE / 'I am program chap2307'.
* External perform
PERFORM EXTFORM IN PROGRAM CHAP2308.

*&---------------------------------------------------------------------*
*& Chapter 23: External perform (called form)
*&---------------------------------------------------------------------*
REPORT CHAP2308.
* Form definition
FORM EXTFORM.
  WRITE / 'I am extform in program chap2308'.
ENDFORM.

*&---------------------------------------------------------------------*
*& Chapter 23: Dynamic external perform (call back form)
*&---------------------------------------------------------------------*
REPORT CHAP2309.
PERFORM EXTFORM IN PROGRAM CHAP2310
       USING 'CALL_BACK_FORM'
            SY-CPROG.
FORM CALL_BACK_FORM.
   WRITE / 'I am the call back form in chap2309.'.
ENDFORM.

*&---------------------------------------------------------------------*
*& Chapter 23: Dynamic external perform
*&---------------------------------------------------------------------*
REPORT CHAP2310.
FORM EXTFORM
    USING F_CALL_BACK_FORM
          F_PROGRAM.
  PERFORM (F_CALL_BACK_FORM) IN PROGRAM (F_PROGRAM).
  WRITE / 'I am the form in chap2310.'.
ENDFORM.

*&---------------------------------------------------------------------*
*& Chapter 24: Working with Field Symbols
*&---------------------------------------------------------------------*
REPORT CHAP2401.
* Defining a Field Symbol
```

```
FIELD-SYMBOLS <FS>.
* Variable for later use
DATA FIELD VALUE 'X'.
* Assigning a field to a Field Symbol
ASSIGN FIELD TO <FS>.
* Using a Field Symbol which has an assigned field
WRITE <FS>.

*&---------------------------------------------------------------------*
*& Chapter 24: Using Field Symbols for variable parts of fields
*&---------------------------------------------------------------------*
REPORT CHAP2402.
DATA: EXTERNAL_RECORD(4000),
      POSITION TYPE I,
      LENGTH TYPE N.
FIELD-SYMBOLS <ENTRY>.
EXTERNAL_RECORD = '0005Smith0007Edwards0005Young'.
DO.
  LENGTH = EXTERNAL_RECORD+POSITION(4).
  IF LENGTH = 0.
    EXIT.
  ENDIF.
  ADD 4 TO POSITION.
  ASSIGN EXTERNAL_RECORD+POSITION(LENGTH) TO <ENTRY>.
  WRITE <ENTRY>.
  ADD LENGTH TO POSITION.
  IF POSITION >= 4000.
    EXIT.
  ENDIF.
ENDDO.

*&---------------------------------------------------------------------*
*& Chapter 24: Using Field Symbols for components of a structure
*&---------------------------------------------------------------------*
REPORT CHAP2403.
* Table work area for later use
TABLES CUSTOMERS.
* Defining a Field Symbol
FIELD-SYMBOLS <OUTPUT>.
* Displaying all fields of all table entries
SELECT * FROM CUSTOMERS.
  NEW-LINE.
  DO.
    ASSIGN COMPONENT SY-INDEX OF STRUCTURE CUSTOMERS TO
<OUTPUT>.
    IF SY-SUBRC <> 0.
```

```
    EXIT.
   ENDIF.
   WRITE <OUTPUT>.
  ENDDO.
ENDSELECT.

*&---------------------------------------------------------------------*
*& Chapter 25: Working with temporary programs
*&---------------------------------------------------------------------*
REPORT CHAP2501.
* Internal table for source code, field for name of temporary program
DATA: SOURCE_TABLE(72) OCCURS 10 WITH HEADER LINE,
      PROGRAM_NAME LIKE SY-CPROG.
* Building the source code
APPEND 'report test.'                    TO SOURCE_TABLE.
APPEND 'form display.'                   TO SOURCE_TABLE.
APPEND 'write "I am a temporary program".' TO SOURCE_TABLE.
APPEND 'endform.'                        TO SOURCE_TABLE.
* Generating the temporary program
GENERATE SUBROUTINE POOL SOURCE_TABLE NAME PROGRAM_NAME.
* Calling a form externally
PERFORM DISPLAY IN PROGRAM (PROGRAM_NAME).

*&---------------------------------------------------------------------*
*& Chapter 25: Syntax errors in temporary programs
*&---------------------------------------------------------------------*
REPORT CHAP2502.
* Variables for later use
DATA: SOURCE_TABLE(72) OCCURS 10 WITH HEADER LINE,
      PROGRAM_NAME LIKE SY-CPROG,
      SYNTAX_CHECK_MESSAGE(128),
      LINE_NO TYPE I.
* Building the source code
APPEND 'report test.'                    TO SOURCE_TABLE.
APPEND 'form display.'                   TO SOURCE_TABLE.
APPEND 'write "I am a temporary program".' TO SOURCE_TABLE.
APPEND 'endform'                         TO SOURCE_TABLE.
* Generating the temporary program, checking syntax errors
GENERATE SUBROUTINE POOL SOURCE_TABLE
                    NAME    PROGRAM_NAME
                    MESSAGE SYNTAX_CHECK_MESSAGE
                    LINE    LINE_NO.
IF SY-SUBRC NE 0.
  WRITE: / 'Syntax error, message', SYNTAX_CHECK_MESSAGE,
         / 'in line', LINE_NO.
  EXIT.
```

```
ENDIF.
* Calling a form externally
PERFORM DISPLAY IN PROGRAM (PROGRAM_NAME).

*&---------------------------------------------------------------------*
*& Chapter 25: A real life example for using a temporary program
*&---------------------------------------------------------------------*
REPORT CHAP2503.
* Variables for later use
PARAMETERS TABNAME(10) DEFAULT 'CUSTOMERS'.
DATA: SOURCE_TABLE(72) OCCURS 100 WITH HEADER LINE,
      PROGRAM_NAME LIKE SY-CPROG,
      SYNTAX_CHECK_MESSAGE(128),
      LINE_NO TYPE I.
* Building the source code
PERFORM BUILD_THE_SOURCE_CODE USING TABNAME.
* Generating the temporary program, checking syntax errors
GENERATE SUBROUTINE POOL SOURCE_TABLE
                    NAME    PROGRAM_NAME
                    MESSAGE SYNTAX_CHECK_MESSAGE
                    LINE    LINE_NO.
IF SY-SUBRC NE 0.
  WRITE: / 'Syntax error, message', SYNTAX_CHECK_MESSAGE,
         / 'in line', LINE_NO.
  EXIT.
ENDIF.
* Calling a form externally
PERFORM DISPLAY_TABLE IN PROGRAM (PROGRAM_NAME).
* Form to build the source code of the temporary program
FORM BUILD_THE_SOURCE_CODE USING F_NAME.
APPEND:
'report ztmpprog.                ' TO SOURCE_TABLE,
'tables                        ' TO SOURCE_TABLE,
       F_NAME                    TO SOURCE_TABLE,
'.                             ' TO SOURCE_TABLE,
'field-symbols <output>.         ' TO SOURCE_TABLE,
'form display_table.             ' TO SOURCE_TABLE,
'select * from                 ' TO SOURCE_TABLE,
            F_NAME               TO SOURCE_TABLE,
'.                             ' TO SOURCE_TABLE,
'  new-line.                     ' TO SOURCE_TABLE,
'  do.                         ' TO SOURCE_TABLE,
'    assign component sy-index     ' TO SOURCE_TABLE,
'          of structure          ' TO SOURCE_TABLE,
           F_NAME                  TO SOURCE_TABLE,
'          to <output>.          ' TO SOURCE_TABLE,
```

```
'    if sy-subrc ne 0. exit. endif.' TO SOURCE_TABLE,
'    write <output>.              ' TO SOURCE_TABLE,
'  enddo.                       ' TO SOURCE_TABLE,
'endselect.                     ' TO SOURCE_TABLE,
'endform.                       ' TO SOURCE_TABLE.
ENDFORM.

*&---------------------------------------------------------------------*
*& Chapter 25: Generating a persistent program
*&---------------------------------------------------------------------*
REPORT CHAP2504.
* Internal table for source code, field for name of temporary program
DATA: SOURCE_TABLE(72) OCCURS 10 WITH HEADER LINE,
      PROGRAM_NAME LIKE SY-CPROG.
* Building the source code
APPEND 'report zgenprog.'                  TO SOURCE_TABLE.
APPEND 'write "I am a generated program".' TO SOURCE_TABLE.
* Insert the report, if necessary
READ REPORT 'zgenprog' INTO SOURCE_TABLE.
IF SY-SUBRC NE 0.
  APPEND 'report zgenprog.'           TO SOURCE_TABLE.
  APPEND 'write "Here is zgenprog".' TO SOURCE_TABLE.
  INSERT REPORT 'zgenprog' FROM SOURCE_TABLE.
ENDIF.
* Execute the report
SUBMIT ZGENPROG AND RETURN.

*&---------------------------------------------------------------------*
*& Chapter 26: Transferring data to a file
*&---------------------------------------------------------------------*
REPORT CHAP2601.
* Data declarations for later use
PARAMETERS FILENAME(128) DEFAULT '/usr/tmp/testfile.dat'
                    LOWER CASE.
TABLES CUSTOMERS.
DATA MSG_TEXT(50).
* Get data for file transfer
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100
                WITH HEADER LINE.
SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS.
SORT ALL_CUSTOMERS BY CITY.
LOOP AT ALL_CUSTOMERS.
  WRITE: / ALL_CUSTOMERS-CITY,
         ALL_CUSTOMERS-NAME.
ENDLOOP.
* Opening the File
```

```
OPEN DATASET FILENAME FOR OUTPUT IN TEXT MODE
                    MESSAGE MSG_TEXT.
IF SY-SUBRC NE 0.
  WRITE: 'File cannot be opened. Reason:', MSG_TEXT.
  EXIT.
ENDIF.
* Transferring Data
LOOP AT ALL_CUSTOMERS.
  TRANSFER ALL_CUSTOMERS-NAME TO FILENAME.
ENDLOOP.
* Closing the File
CLOSE DATASET FILENAME.

*&---------------------------------------------------------------------*
*& Chapter 26: Reading data from a file
*&---------------------------------------------------------------------*
REPORT CHAP2602.
* Data declarations for later use
TABLES CUSTOMERS.
PARAMETERS FILENAME(128) DEFAULT '/usr/tmp/testfile.dat'
                     LOWER CASE.
DATA: MSG_TEXT(50),
      ALL_CUSTOMER_NAMES LIKE CUSTOMERS-NAME OCCURS 100
                     WITH HEADER LINE.
* Opening the File
OPEN DATASET FILENAME FOR INPUT IN TEXT MODE
                    MESSAGE MSG_TEXT.
IF SY-SUBRC NE 0.
  WRITE: 'File cannot be opened. Reason:', MSG_TEXT.
  EXIT.
ENDIF.
* Reading Data
DO.
  READ DATASET FILENAME INTO ALL_CUSTOMER_NAMES.
  IF SY-SUBRC NE 0.
    EXIT.
  ENDIF.
  APPEND ALL_CUSTOMER_NAMES.
ENDDO.
* Closing the file
CLOSE DATASET FILENAME.
* Display the result
LOOP AT ALL_CUSTOMER_NAMES.
  WRITE / ALL_CUSTOMER_NAMES.
ENDLOOP.
```

```
*&---------------------------------------------------------------------*
*& Chapter 26: Transferring data to a file (presentation server)
*&---------------------------------------------------------------------*
REPORT CHAP2603.
* Data declarations for later use
PARAMETERS FILENAME(128) DEFAULT 'c:\users\default\testfile.dat'
                   LOWER CASE.
TABLES CUSTOMERS.
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100
              WITH HEADER LINE.
* Get data for file transfer
SELECT * FROM CUSTOMERS INTO TABLE ALL_CUSTOMERS.
SORT ALL_CUSTOMERS BY CITY.
LOOP AT ALL_CUSTOMERS.
  WRITE: / ALL_CUSTOMERS-CITY,
           ALL_CUSTOMERS-NAME.
ENDLOOP.
* Transferring Data
CALL FUNCTION 'WS_DOWNLOAD'
    EXPORTING
        FILENAME           = FILENAME
    TABLES
        DATA_TAB           = ALL_CUSTOMERS
    EXCEPTIONS
        FILE_OPEN_ERROR    = 1
        OTHERS             = 2.
CASE SY-SUBRC.
  WHEN 1.
    WRITE 'Error when file opened'.
    EXIT.
  WHEN 2.
    WRITE 'Error during data transfer'.
    EXIT.
ENDCASE.

*&---------------------------------------------------------------------*
*& Chapter 26: Reading data from a file (presentation server)
*&---------------------------------------------------------------------*
REPORT CHAP2604.
* Data declarations for later use
PARAMETERS FILENAME(128) DEFAULT 'c:\users\default\testfile.dat'
                   LOWER CASE.
TABLES CUSTOMERS.
DATA ALL_CUSTOMERS LIKE CUSTOMERS OCCURS 100
              WITH HEADER LINE.
CALL FUNCTION 'WS_UPLOAD'
```

```
    EXPORTING
       FILENAME       = FILENAME
    TABLES
       DATA_TAB       = ALL_CUSTOMERS
    EXCEPTIONS
       FILE_OPEN_ERROR = 1
       OTHERS         = 2.
CASE SY-SUBRC.
  WHEN 1.
   WRITE 'Error when file opened'.
   EXIT.
  WHEN 2.
   WRITE 'Error during data transfer'.
   EXIT.
ENDCASE.
* Display the result
LOOP AT ALL_CUSTOMERS.
  WRITE: / ALL_CUSTOMERS-NAME,
         ALL_CUSTOMERS-CITY.
ENDLOOP.

*&---------------------------------------------------------------------*
*& Chapter 28: Sample program for OLE Automation
*&---------------------------------------------------------------------*
REPORT CHAP2801.
* Including OLE types
INCLUDE OLE2INCL.
* Tables and variables for later use
TABLES: CUSTOMERS.
DATA: APPLICATION TYPE OLE2_OBJECT,
      WORKBOOK    TYPE OLE2_OBJECT,
      SHEET       TYPE OLE2_OBJECT,
      CELLS       TYPE OLE2_OBJECT.
* Creating an object
CREATE OBJECT APPLICATION 'excel.application'.
IF SY-SUBRC NE 0.
   WRITE: / 'Error when opening excel.application', SY-MSGLI.
ENDIF.
* Setting properties
SET PROPERTY OF APPLICATION 'Visible' = 1.
* Calling methods
CALL METHOD OF APPLICATION 'Workbooks' = WORKBOOK.
PERFORM ERRORS.
CALL METHOD OF WORKBOOK 'Add'.
PERFORM ERRORS.
CALL METHOD OF APPLICATION 'Worksheets' = SHEET EXPORTING #1 = 1.
```

```
PERFORM ERRORS.
CALL METHOD OF SHEET 'Activate'.
PERFORM ERRORS.
PERFORM FILL_SHEET.

* Subroutine for filling the spread sheet
FORM FILL_SHEET.
  DATA: ROW_MAX TYPE I VALUE 256,
        INDEX   TYPE I.
  FIELD-SYMBOLS: <NAME>.
  SELECT * FROM CUSTOMERS.
    INDEX = ROW_MAX * ( SY-DBCNT - 1 ) + 1.
    DO 4 TIMES.
      ASSIGN COMPONENT SY-INDEX OF STRUCTURE CUSTOMERS TO
<NAME>.
      CALL METHOD OF SHEET 'Cells' = CELLS
                        EXPORTING #1 = INDEX.
      SET PROPERTY OF CELLS 'Value' = <NAME>.
      ADD 1 TO INDEX.
    ENDDO.
  ENDSELECT.
ENDFORM.
* Subroutine for error handling
FORM ERRORS.
  IF SY-SUBRC NE 0.
    WRITE: / 'Error in OLE call', SY-MSGLI.
    EXIT.
  ENDIF.
ENDFORM.
```