**Structures**

**1 Introduction**

💡 Structures are data objects with structured data types (which is a complex data type because it is composed of other data types).

💡 Consist of a sequence of components of any data type, that is, the components of a structure can be, for example, elementary data objects, structures themselves, internal tables or references.

💡 Are used to combine different data objects that belong together. A typical example is an address. It has several components, such as name, street, city, and so on, that belong together.

💡 Play an important role in the context of internal tables and database tables. Structured types serve as line types for these tables. Most internal tables across ABAP programs may have structured line types. For database tables, there is no alternative to structured line types.

💡 Can be created locally in an ABAP program. U can also create them as global DDIC structures (Data Dictionary) in the ABAP Dictionary. Such a DDIC structure defines a globally available structured type (DDIC type).

💡 A [DDIC structure](javascript:call_link('abenddic_structure_glosry.htm')) defines a [structured type](javascript:call_link('abenstructured_type_glosry.htm')) that contains other data types as components. These components can be:

* Elementary data types
* Reference types
* Structured types
* Table types

💡 A DDIC structure that contains other structures as components is called a **nested structure**. Structured components are substructures and substructures can themselves be nested. A structure that contains a string type, reference type, or table type as a direct or nested component is a **deep structure**. A structure that contains only elementary data types (except string types) as direct or nested components is a **flat structure**. A program-internal object declared with reference to a structure is a data object structured correspondingly (or structure for short).

💡 There are other structured types available globally, which may be the structured types most commonly used in ABAP programs like Database tables, CDS Entity...

💡 Database tables defined in the ABAP Dictionary can be used as data types just like DDIC structures in an ABAP program. When this table is activated in DDIC, the SAP system creates a physical copy of its field in the database also. The table is automatically translated into the format that is compatible with the actual database.

💡 Nó được tạo ra để lưu trữ lâu dài thay vì hoạt động runtime như it, được quản lý tập trung, đảm bảo tính toàn vẹn khi dùng để khai báo structure và it. Có thể xem nó như các SSMS tạo table để làm việc với RDBMS, ở đây Database tables được quản lý bởi Database của SAP.

💡 Khi tạo một structure trong chương trình ABAP, có thể sử dụng tên của Database tables để xác định line type của structure này:

TYPES: BEGIN OF customer\_structure,

         customer\_id TYPE zcustomer-customer\_id,

         name        TYPE zcustomer-name,

         city        TYPE zcustomer-city,

       END OF customer\_structure.

💡 Có thể tạo internal table có line type dựa trên cấu trúc của Database tables:

DATA: it\_customers TYPE TABLE OF zcustomer.

💡 A CDS entity such as a CDS view also represents a structured data type and can be used as such in ABAP programs (but not in the ABAP Dictionary). The same applies to DDIC views that are only available in Standard ABAP.

💡 Structures and structured data types can also be defined in the public visibility section of global classes or in global interfaces and then used globally.

**2 Creating Structures and Structured Types**

💡 The typical language elements for creating structures and structured types locally in an ABAP program are BEGIN OF ... END OF .... They are used in combination with the TYPES keyword to create a structured type and the DATA keyword to create a structure. This cheat sheet focuses on locally defined structures and structured types.

**Creating structured types**

* The following statement defines a structured type introduced by TYPES. The type name is preceded by BEGIN OF (which marks the beginning of the structured type definition) and END OF (the end of the definition).
* The components - at least one must be defined - are listed in between.
* Such structured type definitions are usually grouped together in a [chained statement](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenchained_statement_glosry.htm), i.e. TYPES is followed by a colon, and the components are separated by commas.

TYPES: BEGIN OF struc\_type,

         comp1 TYPE ...,

         comp2 TYPE ...,

         comp3 TYPE ...,

         ...,

       END OF struc\_type.

" Alternatively, u can also use the following syntax. However,

" a chained statement may provide better readability.

TYPES BEGIN OF struc\_type.

  TYPES comp1 TYPE ... .

  TYPES comp2 TYPE ... .

  TYPES comp3 TYPE ... .

  ... .

TYPES END OF struc\_type.

* The simplest structures and structured types have elementary components.
* As mentioned above, the components can be of any type, i.e. they can be of structured types themselves, internal table types, or reference types.
* Can use the TYPE and LIKE additions for the types of the components. U can use the LINE OF addition to refer to a table type or an internal table.

TYPES: BEGIN OF struc\_type,

         comp1 TYPE i,                 "elementary type

         comp2 TYPE c LENGTH 5,        "elementary type

         comp3 TYPE structured\_type,   "structured type

         comp4 TYPE itab\_type,         "internal table type

         comp5 TYPE ddic\_type,         "DDIC type

         comp6 TYPE REF TO i,          "data reference

         comp7 LIKE data\_object,       "deriving type from a data object

         comp8 TYPE LINE OF itab\_type, "component has structured type, type derived from internal table type

         comp9 LIKE LINE OF itab,      "component has structured type, type derived from internal table

         comp10,                       "no TYPE/LIKE specification: component is of type c length 1

         ...,

       END OF struc\_type.

💡 Outside of classes, u can also refer to DDIC types using LIKE (... comp11 LIKE ddic\_type, ...). If u actually want to refer to an existing data object, but due to typing errors u inadvertently specify a name that exists as DDIC type, errors may be unavoidable.

**Creating structures**

* To create a structure in an ABAP program, u can use the DATA keyword.
* It works in the same way as the TYPES statement above.
* Unlike the TYPES statement, u can use the [VALUE](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abapdata_options.htm) addition to set default values.

DATA: BEGIN OF struc,

        comp1 TYPE ...,

        comp2 TYPE ... VALUE ...,

        comp3 TYPE i VALUE 99,

        comp4 TYPE i VALUE IS INITIAL,  "Without the addition VALUE, or if IS INITIAL is specified,

                                        "the content is initial.

        comp5 TYPE local\_structured\_type,

        ...,

      END OF struc.

" Alternatively, u can use the following syntax. Similar to above,

"a chained statement may provide better readability.

DATA BEGIN OF struc.

  DATA comp1 TYPE ... .

  DATA comp2 TYPE ... VALUE ... .

... .

DATA END OF struc.

💡 The keywords CLASS-DATA and CONSTANTS can also be used to create structures. In principle, they represent special cases of the general statement shown above.

**CLASS-DATA**

💡 **Cú pháp** [CLASS-DATA attr [options]]. Khai báo một static attributes attr chỉ có thể sử dụng trong phần khai báo của một class hoặc interface. Static attributes có hiệu lực trên class, không phụ thuộc vào các instance của class. Tất cả các thể hiện của class và các class con đều có thể truy cập vào static attributes này.

💡 Static attributes của class cha tồn tại trong tất cả các class con. Static attributes có thể truy cập được từ bên ngoài bằng cách sử dụng class component selector. Sự thay đổi của static attributes ảnh hưởng đến tất cả các class nơi nó tồn tại, độc lập với việc truy cập. Static attributes khai báo bằng CLASS-DATA chỉ có thể truy cập bằng tên class, không thể bằng tên interface.

CLASS c1 DEFINITION.

        PUBLIC SECTION.

          CLASS-DATA text TYPE string VALUE `Static data`.

ENDCLASS.

CLASS exa DEFINITION.

        PUBLIC SECTION.

          CLASS-METHODS main.

ENDCLASS.

CLASS exa IMPLEMENTATION.

        METHOD main.

          cl\_demo\_output=>display\_text( c1=>text ).

        ENDMETHOD.

ENDCLASS.

💡 Static attributes có cấu trúc có thể khai báo dưới dạng static box bằng tùy chọn BOXED.

CLASS lcl\_example DEFINITION.

        PUBLIC SECTION.

          CLASS-DATA: BEGIN OF struct,

                        field1 TYPE i,

                        field2 TYPE c LENGTH 10,

                      END OF struct BOXED.

ENDCLASS.

💡 Ở đây, struct là một thuộc tính tĩnh có cấu trúc, và BOXED là tùy chọn được sử dụng để khai báo nó dưới dạng static box. Khi sử dụng BOXED, hệ thống SAP quản lý bộ nhớ hiệu quả hơn, đặc biệt đối với các cấu trúc dữ liệu lớn.

**CONSTANTS**

💡 CONSTANTS trong ABAP được dùng để khai báo hằng số, giá trị không thay đổi trong suốt thời gian chạy chương trình. Khi khai báo, bắt buộc phải chỉ định giá trị khởi tạo bằng từ khóa VALUE.

💡 Các hằng số này chỉ có thể sử dụng ở các vị trí đọc giá trị và không thể khai báo inline hoặc sử dụng các tùy chọn READ-ONLY, BOXED, và INCLUDE.

CONSTANTS pi TYPE p LENGTH 8 DECIMALS 14

             VALUE '3.14159265358979'.

CONSTANTS: BEGIN OF sap\_ag,

             zip\_code TYPE n LENGTH 5 VALUE '69189',

             city     TYPE string VALUE `Walldorf`,

             country  TYPE string VALUE `Germany`,

           END OF sap\_ag.

CONSTANTS null\_pointer TYPE REF TO object VALUE IS INITIAL.

**Creating structures using existing structured types**

"Local structured type

TYPES: BEGIN OF struc\_type,

         comp1 TYPE i,

         comp2 TYPE c LENGTH 5,

       END OF struc\_type.

"Creating a structure using a local structured type

DATA struc\_1 TYPE struc\_type.

"Creating structures based on globally available types from the DDIC

"Note: When referring to such types, u cannot provide start values for the individual components.

DATA: struc\_2 TYPE some\_ddic\_structure,

      struc\_3 TYPE some\_ddic\_table,

      struc\_4 TYPE some\_cds\_view.

" Tạo một biến có kiểu dữ liệu từ một public structure trong một global class

DATA struc\_5 TYPE cl\_some\_class=>struc\_type.

DATA: struc\_6 LIKE struc\_1, " Tạo một biến có cùng kiểu dữ liệu với biến struc\_1

      struc\_7 LIKE LINE OF some\_itab, " Tạo structure cùng kiểu dữ liệu với một dòng của it

      struc\_8 TYPE LINE OF some\_itab\_type. " Tạo structure cùng kiểu dữ liệu với một it types

**Creating structures by inline declaration**

Điều này đặc biệt hữu ích khi cần khai báo các đối tượng dữ liệu ngay tại vị trí sử dụng chúng (operand positions).

"Structures created inline instead of an extra declared variable

DATA struc\_9 LIKE struc\_1.

struc\_9 = struc\_1

"Type is derived from the right-hand structure; the content of struc is assigned, too.

DATA(struc\_10) = struc\_1.

FINAL(struc\_11) = struc\_9.

"Using the VALUE operator

"A structure declaration as follows (without providing component

"value assignments) ...

DATA(struc\_a) = VALUE struc\_type( ).

"... is similar to the following declaration.

DATA struc\_b TYPE struc\_type.

"Structures declared inline instead of an extra declared variable

"Example: SELECT statement

"Extra declaration

DATA struc\_12 TYPE zdemo\_abap\_fli.

SELECT SINGLE \*

  FROM zdemo\_abap\_fli

  WHERE carrid = 'LH'

  INTO @struc\_12.

"Inline declaration

SELECT SINGLE \*

  FROM zdemo\_abap\_fli

  WHERE carrid = 'LH'

  INTO @DATA(struc\_13).

"Example: Loop over an internal table

DATA itab TYPE TABLE OF zdemo\_abap\_fli WITH EMPTY KEY.

... "itab is filled

"Extra declaration

DATA wa\_1 LIKE LINE OF itab.

LOOP AT itab INTO wa\_1.

  ...

ENDLOOP.

"Inline declaration

LOOP AT itab INTO DATA(wa\_2).

  ...

ENDLOOP.

**Anonymous Structures**

Using the instance operator [NEW](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenconstructor_expression_new.htm) and [CREATE DATA](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abapcreate_data.htm) statements, u can create [anonymous data objects](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenanonymous_data_object_glosry.htm), such as anonymous structures. The NEW addition of the INTO clause of an ABAP SQL SELECT statement also creates an anonymous data object. As outlined below, u can access the components or the entire data objects by [dereferencing](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abendereferencing_operat_glosry.htm). For more information, refer to the [Dynamic Programming](https://github.com/SAP-samples/abap-cheat-sheets/blob/main/06_Dynamic_Programming.md) and [Constructor Expressions](https://github.com/SAP-samples/abap-cheat-sheets/blob/main/05_Constructor_Expressions.md) cheat sheets.

"Without assigning component values in the parentheses, the anonymous

"structure is initial.

DATA(struc\_ref\_a) = NEW struc\_type( ).

DATA struc\_ref\_b TYPE REF TO DATA.

struc\_ref\_b = NEW struc\_type( ).

"Multiple syntax options are available for CREATE DATA

"statements. See the cheat sheets mentioned.

CREATE DATA struc\_ref\_b TYPE struc\_type.

DATA struc\_ref\_c TYPE REF TO struc\_type.

"Implicit data type definition

CREATE DATA struc\_ref\_c.

"NEW addition of the INTO clause of an ABAP SQL SELECT statement

SELECT SINGLE carrid, carrname

 FROM zdemo\_abap\_carr

 WHERE carrid = char`LH`

 INTO NEW @DATA(struc\_ref\_d).

[**CREATE DATA**](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abapcreate_data.htm)

CREATE DATA trong ABAP được sử dụng để tạo các anonymous data objects và gán tham chiếu của đối tượng đó cho một data reference variable.

**3 Variants of Structures**

Depending on the component type, the structure can be a [flat structure](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenflat_structure_glosry.htm), a [nested structure](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abennested_structure_glosry.htm), or a [deep structure](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abendeep_structure_glosry.htm).

* **Flat structures** contain only elementary types that have a fixed length, that is, there are no internal tables, reference types or strings as components. Nesting does not matter in this context. Even a nested structure is considered flat unless a substructure contains a deep component.

DATA: BEGIN OF struc,

        comp1 TYPE i,

        comp2 TYPE c LENGTH 15,

        comp3 TYPE p LENGTH 8 DECIMALS 2,

        ...,

      END OF struc.

* Nested structures: At least one component of a structure is a substructure, that is, it refers to another structure. The following example has multiple substructures.

DATA: BEGIN OF address\_n,

        BEGIN OF name,

          title   TYPE string VALUE `Mr.`,

          prename TYPE string VALUE `Duncan`,

          surname TYPE string VALUE `Pea`,

        END OF name,

        BEGIN OF street,

          name TYPE string VALUE `Vegetable Lane`,

          num  TYPE string VALUE `11`,

        END OF street,

        BEGIN OF city,

          zipcode TYPE string VALUE `349875`,

          name    TYPE string VALUE `Botanica`,

        END OF city,

    END OF address\_n.

* Deep structures: Contains at least one internal table, reference type, or string as a component.

DATA: BEGIN OF address\_d,

        name    TYPE string VALUE `Mr. Duncan Pea`,

        street  TYPE string VALUE `Vegetable Lane 11`,

        city    TYPE string VALUE `349875 Botanica`,

        details TYPE TABLE OF some\_table WITH EMPTY KEY,

      END OF address\_d.

* Although the following structure looks quite simple, it is not a flat structure, but a deep structure, because it contains strings.

DATA: BEGIN OF address,

        name   TYPE string VALUE `Mr. Duncan Pea`,

        street TYPE string VALUE `Vegetable Lane 11`,

        city   TYPE string VALUE `349875 Botanica`,

      END OF address.

**💡 Note**

* The data types of DDIC types are all flat (not nested) structures. Exception: Components of type string can be contained.
* [Work areas](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenwork_area_glosry.htm) of ABAP SQL statements cannot contain any deep components other than strings among others.
* Especially for assignments and comparisons of deep structures, the compatibility of the source and target structure must be taken into account.

**4 Accessing (Components of) Structures**

💡 Structures can be accessed as a whole. U can also address the individual components of structures at the appropriate operand positions.

💡 To address the components, use the structure component selector -.

💡 For variables with reference to a structured data object, the object component selector -> can be used: ...dref->comp .... The following syntax also works, but is less convenient: ... dref->\*-comp ....

💡 ADT and the ABAP Editor provide code completion for structure components after the component selectors.

"Addressing components via the structure component selector

... struc-comp1 ...

... struc-comp2 ...

... struc-comp3 ...

"Examples for addressing the whole structure and individual components

IF struc IS INITIAL.

  ...

ENDIF.

IF struc-comp1 = 1.

  ...

ENDIF.

DATA(complete\_struc) = struc.

DATA(comp\_value) = struc-comp2.

"Type and data declarations

TYPES: type\_1 TYPE structured\_type-comp1,

       type\_2 LIKE struc-comp1.

DATA: var\_1 TYPE structured\_type-comp1,

      var\_2 LIKE struc-comp1.

"Variables with reference to a structured data object

DATA ref\_struc\_1 TYPE REF TO structured\_type.

ref\_struc\_1 = NEW #( ).

"Excursion: Creating a reference variable using inline declaration

DATA(ref\_struc\_2) = NEW structured\_type( ).

... ref\_struc\_1->comp1 ...

... ref\_struc\_1->\*-comp1 ...  "Using the dereferencing operator

... ref\_struc\_2->comp2 ...

... ref\_struc\_2->\*-comp2 ...  "Using the dereferencing operator

💡 Nested components can be addressed using chaining:

... struc-substructure-comp1 ...

... address\_n-name-title ...

💡 There are syntax options for dynamically accessing structure components. See the Dynamic Porgramming cheat sheet.

**5 Populating Structures**

💡 Can copy the content of a structure to another using the assignment operator =. In the following example, it is assumed that the target and source structures are of compatible types. In general, note that special conversion and comparison rules apply to value assignments involving structures.

some\_struc = another\_struc.

"When creating a new structure by inline declaration, the type of

"the right-hand structure is derived and the content is assigned.

DATA(struc\_inl) = some\_struc.

💡 To assign values to individual structure components, use the component selector.

TYPES: BEGIN OF addr\_struc,

        name   TYPE string,

        street TYPE string,

        city   TYPE string,

       END OF addr\_struc.

DATA address TYPE addr\_struc.

address-name   = `Mr. Duncan Pea`.

address-street = `Vegetable Lane 11`.

address-city   = `349875 Botanica`.

**5.1 Using the VALUE Operator**

* The [VALUE](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenconstructor_expression_value.htm) operator can be used to construct the content of complex data objects such as structures or internal tables.
* It is particularly useful because assigning values by addressing the structure components individually can be very cumbersome, especially when assigning values to structure components at the [operand position](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenoperand_position_glosry.htm).
* If the type of the operand can be inferred implicitly, the # character can be used used before the parentheses. Otherwise, the type must be specified explicitly.
* The VALUE operator and inline declarations can be used to create and populate structures in one go.
* Note that there are special [conversion](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenconversion_struc.htm) and [comparison](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenlogexp_rules_operands_struc.htm) rules for structures. See the ABAP Keyword Documentation for more details.

"# used: type of the operand can be implicitly derived

address = VALUE #( name   = `Mr. Duncan Pea`

                   street = `Vegetable Lane 11`

                   city   = `349875 Botanica` ).

"Declaring a structure inline

"Type used explicitly: type of the operand cannot be implicitly derived

DATA(addr) = VALUE addr\_struc( name   = `Mr. Duncan Pea`

                               street = `Vegetable Lane 11`

                               city   = `349875 Botanica` ).

"Using the BASE addition to retain existing component values

addr = VALUE #( BASE addr street = `Some Street 1` ).

\*NAME              STREET           CITY

\*Mr. Duncan Pea    Some Street 1    349875 Botanica

"Without the BASE addition, the components are initialized

addr = VALUE #( street = `Another Street 2` ).

\*NAME       STREET              CITY

\*           Another Street 2

"Nesting value operators

TYPES: BEGIN OF struc\_nested,

        a TYPE i,

        BEGIN OF nested\_1,

          b TYPE i,

          c TYPE i,

        END OF nested\_1,

        BEGIN OF nested\_2,

          d TYPE i,

          e TYPE i,

        END OF nested\_2,

      END OF struc\_nested.

DATA str\_1 TYPE struc\_nested.

str\_1 = VALUE #( a        = 1

                 nested\_1 = VALUE #( b = 2 c = 3 )

                 nested\_2 = VALUE #( d = 4 e = 5 ) ).

"Inline declaration

"Component a is not specified here, i.e. its value remains initial.

DATA(str\_2) = VALUE struc\_nested( nested\_1 = VALUE #( b = 2 c = 3 )

                                  nested\_2 = VALUE #( d = 4 e = 5 ) ).

"Apart from the VALUE operator, the NEW operator can be used to create

"a data reference variable (and populate the structure)

DATA(str\_ref) = NEW struc\_nested( a        = 1

                                  nested\_1 = VALUE #( b = 2 c = 3 )

                                  nested\_2 = VALUE #( d = 4 e = 5 ) ).

**5.2 Using the NEW Operator**

💡 Using the instance operator NEW, u can create anonymous data objects, such as anonymous structures. U can access the components or the entire data objects by dereferencing. For more information, refer to the Dynamic Programming and Constructor Expressions cheat sheets.

"Creating a data reference variable

DATA addr\_ref1 TYPE REF TO addr\_struc.

"Populating the anonymous structure

addr\_ref1 = NEW #( name   = `Mr. Duncan Pea`

                   street = `Vegetable Lane 11`

                   city   = `349875 Botanica` ).

addr\_ref1->name = `Mrs. Jane Doe`.

"Declaring an anonymous structure/a data reference variable inline

DATA(addr\_ref2) = NEW addr\_struc( name   = `Mr. Duncan Pea`

                                  street = `Vegetable Lane 11`

                                  city   = `349875 Botanica` ).

addr\_ref2->\* = VALUE #( BASE addr\_ref2->\* name = `Mr. John Doe` ).

**5.3 Using the CORRESPONDING Operator and MOVE-CORRESPONDING Statements**

💡 Can use statements with MOVE-CORRESPONDING and the CORRESPONDING operator to assign values to structure components, especially when assigning values from a source structure to a target structure which have incompatible types and/or **differently named components**.

💡 Both are used to assign identically named components of structures to each other. The syntax also works for structures of the same type. Also note the special conversion and comparison rules for structures in this context.

**💡 Note**

* The [CL\_ABAP\_CORRESPONDING](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abencl_abap_corresponding.htm) system class is available for making assignments. See the ABAP Keyword Documentation for the details.
* The INTO clause of ABAP SQL statements has the CORRESPONDING addition. There, the following basic rule applies, which affects the value assignment: Without the CORRESPONDING ... addition, column names do not matter, only the position. With the CORRESPONDING ... addition, the position of the columns does not matter, only the name. See examples in the ABAP SQL cheat sheet.

💡 The following examples demonstrate the value assignment using MOVE-CORRESPONDING statements and the CORRESPONDING operator with various additions. The focus is on flat structures only.

"Moves identically named components; content in other components

"of the targets structure are kept.

MOVE-CORRESPONDING struc TO diff\_struc.

"Initializes target structure; moves identically named components

diff\_struc = CORRESPONDING #( struc ).

"Same effect as the first MOVE-CORRESPONDING statement;

"addition BASE keeps existing content

diff\_struc = CORRESPONDING #( BASE ( diff\_struc ) struc ).

"MAPPING addition: Specifying components of a source structure that are

"assigned to the components of a target structure in mapping

"relationships.

diff\_struc = CORRESPONDING #( BASE ( diff\_struc ) struc MAPPING comp1 = compa ).

"EXCEPT addition: Excluding components from the assignment.

diff\_struc = CORRESPONDING #( BASE ( diff\_struc ) struc EXCEPT comp1 ).

💡 Value assignments in deep structures

* In the context of deep structures, there are additional syntax variants available for [MOVE-CORRESPONDING](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abapmove-corresponding.htm) statements and the [CORRESPONDING](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenconstructor_expr_corresponding.htm) operator.
* The following examples focus on internal tables as structure components. Check out the syntax in action in the executable example.

"Nonidentical elementary component types are kept in target

"structure which is true for the below MOVE-CORRESPONDING statements;

"existing internal table content is replaced by content of

"the source table irrespective of identically named components

MOVE-CORRESPONDING deep\_struc TO diff\_deep\_struc.

"Existing internal table content is replaced but the value

"assignment happens for identically named components only.

MOVE-CORRESPONDING deep\_struc TO diff\_deep\_struc EXPANDING NESTED TABLES.

"Existing internal table content is kept; table content of the source

"structure are added but the value assignment happens like the first

"MOVE-CORRESPONDING statement without further syntax additions.

MOVE-CORRESPONDING deep\_struc TO diff\_deep\_struc KEEPING TARGET LINES.

"Existing internal table content is kept; table content of the source

"structure are added; the value assignment happens like the statement

"MOVE-CORRESPONDING ... EXPANDING NESTED TABLES.

MOVE-CORRESPONDING deep\_struc TO diff\_deep\_struc EXPANDING NESTED TABLES KEEPING TARGET LINES.

"Target structure is initialized; the value assignment for an internal

"table happens irrespective of identically named components.

diff\_deep\_struc = CORRESPONDING #( deep\_struc ).

"Target structure is initialized; the value assignment for an internal

"table happens for identically named components only.

diff\_deep\_struc = CORRESPONDING #( DEEP deep\_struc ).

"Nonidentical elementary component types are kept in target structure;

"internal table content is replaced; there, the value assignment

"happens like using the CORRESPONDING operator without addition.

diff\_deep\_struc = CORRESPONDING #( BASE ( diff\_struc ) deep\_struc ).

"Nonidentical elementary component types are kept in target structure;

"internal table content is replaced; there, the value assignment

"happens like using the CORRESPONDING operator with the addition DEEP.

diff\_deep\_struc = CORRESPONDING #( DEEP BASE ( diff\_struc ) deep\_struc ).

"Nonidentical elementary component types are kept in target structure;

"internal table content is kept, too, and table content of the

"source structure are added; there, the value assignment

"happens like using the CORRESPONDING operator without addition.

diff\_deep\_struc = CORRESPONDING #( APPENDING BASE ( diff\_struc ) deep\_struc ).

"Nonidentical elementary component types are kept in target structure;

"internal table content is kept, too, and table content of the

"source structure are added; there, the value assignment

"happens like using the CORRESPONDING operator with the addition DEEP.

diff\_deep\_struc = CORRESPONDING #( DEEP APPENDING BASE ( diff\_struc ) deep\_struc ).

**6 Clearing Structures**

💡 Can reset individual components to their initial values and clear the entire structure using the CLEAR keyword.

CLEAR struc-component.

CLEAR struc.

"Note: An assignment using the VALUE operator without entries in the parentheses clears the structure.

struc = VALUE #( ).

"The same applies to data reference variables pointing to structures.

struc\_ref = NEW #( ).

**7 Processing Structures**

💡 Structures are primarily used to process data from tables. In this context, structures often take on the role of a [work area](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenwork_area_glosry.htm). The following code snippets cover only a selection. For more examples, see the cheat sheets about internal tables and ABAP SQL.

💡 **Reading a row from a database table into a structure that has a compatible type**. Note that, since database tables are flat, the target structure must also be flat. In the example below, the [SINGLE](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abapselect_single.htm) addition reads only a single row into the structure. It returns the first entry that matches the WHERE condition.

"Creating a structure with a compatible type

DATA ls\_fli1 TYPE zdemo\_abap\_fli.

SELECT SINGLE FROM zdemo\_abap\_fli

  FIELDS \*

  WHERE carrid = 'LH'

  INTO @ls\_fli1.

"Target structure declared inline

SELECT SINGLE FROM zdemo\_abap\_fli

  FIELDS \*

  WHERE carrid = 'LH'

  INTO @DATA(ls\_fli2).

💡 **Reading a row from a database table into a structure that has an incompatible type**. Components in the structure with identical names are filled.

SELECT SINGLE FROM zdemo\_abap\_fli

  FIELDS \*

  WHERE carrid = 'AA'

  INTO CORRESPONDING FIELDS OF @ls\_fli\_diff.

💡 **Reading a line from an internal table into a structure** ...

... using a SELECT statement. Note the specified alias name and that ABAP variables like internal tables must be escaped with @. The addition INTO CORRESPONDING FIELDS OF also applies here.

SELECT SINGLE FROM @itab AS itab\_alias

  FIELDS \*

  WHERE ...

  INTO @DATA(ls\_struc).

  "INTO CORRESPONDING FIELDS OF @some\_existing\_struc.

💡... using a READ TABLE statement. The code snippet below shows the reading of a line into a [work area](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenwork_area_glosry.htm), a [field symbol](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenfield_symbol_glosry.htm), and a [data reference variable](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abendata_reference_variable_glosry.htm), all of which represent structured data objects that are declared inline. In the following example, a line is read based on the line number by specifying INDEX. For more details, see the section *Determining the target area* in the cheat sheet [Internal Tables](https://github.com/SAP-samples/abap-cheat-sheets/blob/main/01_Internal_Tables.md).

READ TABLE itab INTO DATA(wa) INDEX 1.

READ TABLE itab ASSIGNING FIELD-SYMBOL(<fs>) INDEX 2.

READ TABLE itab REFERENCE INTO DATA(dref) INDEX 3.

💡... using a [table expression](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abentable_expression_glosry.htm). The code snippet shows how to read a line into a structure declared inline. The index is given in square brackets.

DATA(ls\_table\_exp) = itab[ 3 ].

💡 **Sequentially reading** ...

... a row from a database table into a structure. A SELECT loop can be specified with the syntax SELECT ... ENDSELECT.. In the following example, the row found and returned in a structure declared inline can be processed further.

SELECT FROM zdemo\_abap\_fli

  FIELDS \*

  WHERE carrid = 'AZ'

  INTO @DATA(ls\_sel\_loop).

  IF sy-subrc = 0.

    ...

  ENDIF.

ENDSELECT.

💡... a line from an internal table into a structure using a LOOP AT statement. There are many ways to specify the condition on which the loop is based. The following example covers the option of reading all lines sequentially into a field symbol declared inline. When using a field symbol, you can, for example, directly modify components.

LOOP AT itab ASSIGNING FIELD-SYMBOL(<fs>).

  <fs>-comp1 = ...

  ...

ENDLOOP.

💡 Inserting a single row into a database table from a structure using ABAP SQL statements with INSERT. The following statements can be considered as alternatives. The third statement shows that instead of inserting a row from an existing structure, you can create and fill a structure directly. Note that you should avoid inserting a row with a particular key into the database table if a row with the same key already exists. Note that with this and the followig syntax, various options/expressions are possible.

INSERT INTO dbtab VALUES @struc.

INSERT dbtab FROM @struc.

INSERT dbtab FROM @( VALUE #( comp1 = ... comp2 = ... ) ).

💡 Updating a single row in a database table from a structure using ABAP SQL statements with UPDATE. Note that this syntax changes the entire row and all of its components.

UPDATE dbtab FROM @struc.

UPDATE dbtab FROM @( VALUE #( comp1 = ... comp2 = ... ) ).

💡 If you want to update a database table row from a structure by specifying components to be changed without overwriting other components, you can use the following method. First, read the desired row from the database table into a structure. Then, use the VALUE operator with the BASE addition and specify the components to be changed.

SELECT SINGLE \*

    FROM dbtab

    WHERE ...

    INTO @DATA(wa).

UPDATE dbtab FROM @( VALUE #( BASE wa comp2 = ... comp4 = ... ) ).

💡 Updating or creating a single row in a database table from a structure using ABAP SQL statements with MODIFY. If a row with the same key as specified in the structure already exists in the database table, the row is updated. If no row with the keys specified in the structure exists, a new row is created in the database table.

MODIFY dbtab FROM @struc.

MODIFY dbtab FROM @( VALUE #( comp1 = ... comp2 = ... ) ).

💡 **Adding lines to and updating single lines in an internal table from a structure** using INSERT, APPEND, and MODIFY statements.

* Note that all statements, including INSERT and MODIFY, are ABAP statements in this context, not ABAP SQL statements.
* Both INSERT and APPEND add one or more lines to an internal table. While APPEND adds at the bottom of the internal table, INSERT can be used to add lines at a specific position in the table. If you do not specify the position, the lines are also added at the bottom of the table. However, unlike APPEND, INSERT does not set sy-tabix.
* MODIFY changes the content of an internal table entry.
* Statements using the VALUE operator to directly create and populate the structures are also possible. For more information and code snippets, see the [Internal Tables](https://github.com/SAP-samples/abap-cheat-sheets/blob/main/01_Internal_Tables.md) cheat sheet.

INSERT struc INTO TABLE itab.

APPEND struc TO itab.

MODIFY TABLE itab FROM struc.

**8 Excursions**

**8.1 Including Structures**

* [INCLUDE TYPE](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abapinclude_type.htm) and [INCLUDE STRUCTURE](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abapinclude_type.htm) statements are used in the context of local structures.
* Structured data objects and types created with ... BEGIN OF... END OF ... can use this syntax to include components of another structure, whether it is a locally defined or global structure, without creating substructures.
* INCLUDE TYPE can be used to include a structured type.
* You can use INCLUDE STRUCTURE to include a structure.

**💡 Note**

* They are not additions of ... BEGIN OF ... END OF ... but individual ABAP statements.
* If you use a chained statement with a colon to declare the structure, the inclusion of other structures with these statements interrupts the chained statement, that is, the components of the included structures are included as direct components of the [superstructure](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abensuperstructure_glosry.htm).
* By using the optional AS addition and specifying a name, the included components can be addressed by this common name as if they were actually components of a substructure.
* The optional RENAMING WITH SUFFIX addition, followed by a name, gives the included components a suffix name to avoid naming conflicts with other components.

💡 The following example shows how structured types and data objects are included in another structure. First, three structured types and a structured data object based on one of these types are created. Then, the types and the structure are included in the structured type address\_type. The executable example demonstrates a structure that includes other structures in this way.

TYPES: BEGIN OF name\_type,

        title   TYPE string,

        prename TYPE string,

        surname TYPE string,

      END OF name\_type,

      BEGIN OF street\_type,

        name TYPE string,

        num  TYPE string,

      END OF street\_type,

      BEGIN OF city\_type,

        zipcode TYPE string,

        name    TYPE string,

      END OF city\_type.

DATA city\_struc TYPE city\_type.

TYPES BEGIN OF address\_type.

      INCLUDE TYPE name\_type AS name.

      INCLUDE TYPE street\_type AS street RENAMING WITH SUFFIX \_street. " Thêm hậu tố

      INCLUDE STRUCTURE city\_struc AS city RENAMING WITH SUFFIX \_city.

TYPES END OF address\_type.

**8.2 Getting Structured Type Information and Creating Structures at Runtime**

💡 Using [Runtime Type Services (RTTS)](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenrun_time_type_services_glosry.htm) you can ...

* get type information on data objects, data types or [instances](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abeninstance_glosry.htm) at runtime ([Runtime Type Identification (RTTI)](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenrun_time_type_identific_glosry.htm)).
* define and create new data types as [type description objects](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abentype_object_glosry.htm) at runtime ([Runtime Type Creation (RTTC)](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenrun_time_type_creation_glosry.htm)).

For more information, see the [Dynamic Programming](https://github.com/SAP-samples/abap-cheat-sheets/blob/main/06_Dynamic_Programming.md) cheat sheet.

**9 Executable Example**

[zcl\_demo\_abap\_structures](https://github.com/SAP-samples/abap-cheat-sheets/blob/main/src/zcl_demo_abap_structures.clas.abap)

**💡 Note**

* The executable example covers the following topics, among others:
  + Creating structures and structured types
  + Variants of structures
  + Accessing, populating, and clearing structures
  + Structures in the context of tables
  + Including structures
* The steps to import and run the code are outlined [here](https://github.com/SAP-samples/abap-cheat-sheets/blob/main/README.md#-getting-started-with-the-examples).
* [Disclaimer](https://github.com/SAP-samples/abap-cheat-sheets/blob/main/README.md#%EF%B8%8F-disclaimer)