[Содержание](#Содержание)

Динамическое программирование

[FIELD-SYMBOLS](#FIELD_SYMBOLS)

## [Обобщенные типы](#Обобщенные_типы)

[CASTING](#CASTING)

[#](#диез)

[Процедура de-referencing /разыменования/](#de_referencing)

[CREATE DATA](#CREATE_DATA)

**INTO CORRESPONDING FIELDS OF TABLE**

SELECT \* FROM <DATABASE TABLE>

INTO CORRESPONDING FIELDS OF TABLE <INTERNAL TABLE> .

**As per SAP standards *select into corresponding statement is not advisable* as it affects the performance of an application *because it has to compare each field with database*.**

**host variables**

The escape character can only be used in programs, in which the program property [*fixed point arithmetic*](javascript:call_link('abenfixed_point_arithmetic_glosry.htm')) is activated. When the escape character is used, the syntax check is performed in a [*strict mode*](javascript:call_link('abenopensql_strict_mode_740_sp05.htm')), which handles the statement more strictly than the regular syntax check.

DATA carrid TYPE spfli-carrid.

cl\_demo\_input=>request( CHANGING field = carrid ).

DATA: BEGIN OF *result*,

         carrid TYPE sflight-carrid,

         connid TYPE sflight-connid,

       END OF result.

SELECT carrid, connid FROM sflight INTO CORRESPONDING FIELDS OF *@result* WHERE carrid = @carrid.

  ...

ENDSELECT.

FIELD-SYMBOLS *<result>* LIKE result.

SELECT carrid, connid FROM sflight INTO CORRESPONDING FIELDS OF *@<result>*

WHERE carrid = @carrid.

  ...

ENDSELECT.

DATA *dref* LIKE REF TO result.

SELECT carrid, connid FROM sflight INTO CORRESPONDING FIELDS OF *@dref->\** WHERE carrid =@carrid.

  ...

ENDSELECT.

**SY-STEPL** - the index of the screen table row

The system variable *SY-STEPL* to find out the *index of the screen table row that is currently being processed*. The system sets this variable each time through the loop. SY-STEPL always has values from 1 to the number of rows currently displayed. This is useful when you are transferring field values back and forth between a screen table and an internal table.

You can declare a *table-offset variable* in your program /often called *BASE*, and usually initialized with SY-LOOPC/ and use it with SY-STEPL to get the internal table row that corresponds to the current screen table row.

*\*SCREEN FLOW LOGIC*

PROCESS BEFORE OUTPUT.

LOOP.

MODULE READ\_INTTAB.

ENDLOOP.

PROCESS AFTER INPUT.

LOOP.

MODULE MODIFY\_INTTAB.

ENDLOOP.

*\*ABAP MODULES*

MODULE READ\_INTTAB.

IND = BASE + SY-STEPL - 1.

READ TABLE INTTAB INDEX IND.

ENDMODULE.

MODULE MODIFY\_INTTAB.

IND = BASE + SY-STEPL - 1.

MODIFY INTTAB INDEX IND.

ENDMODULE.

**tableview-top\_line** -scrolling variable

If MOVE-CORRESPONDING is used, the operands WA-RATES and IT-RATES must be both structures or both internal tables.

**LOOP - WITH CONTROL**

LOOP [AT itab [INTO wa] [CURSOR top\_line] [FROM n1] [TO n2]] WITH CONTROL contrl.

  ...

ENDLOOP.

The loop sequentially processes the visible rows of the table control contrl by executing one loop pass for reach table control row.

The statement block between LOOP and ENDLOOP can contain the keywords [FIELD](javascript:call_link('dynpfield.htm')), [MODULE](javascript:call_link('dynpmodule.htm')), [CHAIN](javascript:call_link('dynpchain.htm')), and [ENDCHAIN](javascript:call_link('dynpendchain.htm')) from the flow logic. Nesting of loops is not possible.

PROCESS BEFORE OUTPUT.   
  MODULE prepare\_tab.   
  LOOP *AT spfli\_tab INTO spfli* WITH CONTROL *flight\_tab*.   
  ENDLOOP.

PROCESS AFTER INPUT.   
  LOOP AT spfli\_tab.   
    MODULE modify\_tab.   
  ENDLOOP.

**Динамическое программирование в ABAP**

<https://abap-blog.ru/osnovy-abap/dinamicheskoe-programmirovanie/dinamicheskoe-programmirovanie-v-abap/>

Курс BC402.

**FIELD-SYMBOLS**

[Содержание](#Содержание)

***IS ASSIGNED***

*FIELD-SYMBOLS* <gfs\_any\_string> TYPE CSEQUENCE. *“csequence - c, string*

DATA: gv\_string TYPE string VALUE 'TEST'.

ASSIGN gv\_string TO <gfs\_any\_string>.

IF <gfs\_any\_string> *IS ASSIGNED*.

  WRITE <gfs\_any\_string>.

ELSE.

  WRITE 'Ссылка на переменную не присвоена!!'.

ENDIF.

***READ TABLE … ASSIGNING …***

READ TABLE gt\_tab INDEX 1 *ASSIGNING* <gfs\_any\_struct>.

LOOP AT gt\_tab *ASSIGNING* <gfs\_any\_struct>.

ENDLOOP.

Применяя field-symbols при обработке больших таблиц /особенно в циклах/, можно получить существенную оптимизацию в работе программы, т.к. системе не нужно каждый раз копировать данные из таблицы в переменную.

***ASSIGN COMPONENT***

Когда структура для field-symbol является неопределенной /тип any/, то значение компонента можно получить, используя оператор – ASSIGN COMPONENT.

DATA: ls\_flight TYPE sflight.

FIELD-SYMBOLS: *<lfs\_flight>* TYPE *ANY*,

               <connid>     TYPE sflight-connid,

               <seatsocc>   TYPE sflight-seatsocc.

ls\_flight-connid = '182S'.

ls\_flight-seatsocc = 60.

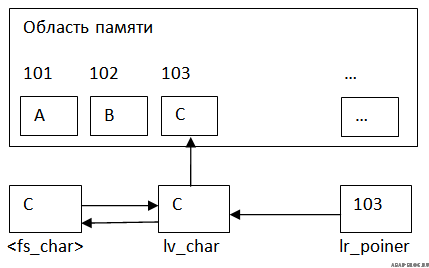
ASSIGN ls\_flight TO <lfs\_flight>.

*ASSIGN COMPONENT* 'CONNID' OF STRUCTURE ls\_flight TO <connid>.

ASSIGN COMPONENT 'SEATSOCC' OF STRUCTURE ls\_flight TO <seatsocc>.

**Отличие *field-symbol* от *ссылки***

[Содержание](#Содержание)



У нас есть три объекта

1. ***lv\_char*** – переменная которая имеет символьный тип, хранится в блоке памяти 103 и имеет значение *С*.
2. ***<fs\_char>*** – field-symbol - *псевдоним для lv\_char*.  Если изменить одну переменную, это изменение так же отразится на значении другой и наоборот.
3. ***lr\_pointer*** – ссылочная переменная, которая *содержит указатель на адрес в памяти* – 103, где хранится значение переменной lv\_char. Изменение значения переменной lv\_char никак не отражается на переменной lr\_pointer т.к. она хранит лишь адрес на блок памяти. Однако можно *с помощью операции разыменования изменить переменную lv\_char через lr\_po*inter.

## **Обобщенные типы**

[Содержание](#Содержание)

|  |  |
| --- | --- |
| **Тип** | **Описание** |
| ***any*** | Любой тип данных |
| ***any table*** | Любая внутренняя таблица |
| ***clike*** | Обобщенный символьный тип - **c**, **d**, **n**, **t**, **string**, а так же плоские структуры, состоящие из элементов символьных типов |
| ***csequence*** | Текстовая последовательность - **c**, **string** |
| ***data*** | Любой тип данных - *аналогично* ***any*** *в случае объявления TYPE data;* если объявлять TYPE REF TO DATA, будут подразумеваться ссылки на данные, но не объектные ссылки. Данный тип может быть использован в ссылочных переменных. |
| ***decfloat*** | Числовой тип с плавающей запятой - **decfloat16**, **decfloat34.** |
| ***hashed table*** | Любая хеш таблица |
| ***index table*** | Любая стандартная или сортированная внутренняя таблица. |
| ***numeric*** | Числовой тип - **i** (**b**, **s**), **p**, **decfloat16**, **decfloat34**, **f** |
| ***object*** | Любой объектный тип |
| ***simple*** | Любой элементарный тип данных, включая плоские структуры состоящие из символьных элементов. |
| ***sorted table*** | Любая сортированная таблица |
| ***standard table*** | Любая стандартная таблица |
| ***table*** | Аналогично предыдущему |
| **xsequence** | Байтовая последовательность - **x**, **xstring** |

**CASTING**

[Содержание](#Содержание)

В момент присвоения, с помощью дополнения CASTING, возможна адаптация значений, если типы данных не совпадают, но при этом они совместимы.

1.

DATA: BEGIN OF gs\_test,

char1 VALUE '1',

    char2 VALUE '2',

       END OF gs\_test.

TYPES: ty\_num\_2 TYPE n LENGTH 2.

FIELD-SYMBOLS: <fs\_int> TYPE ty\_num\_2.

ASSIGN gs\_test TO <fs\_int> *CASTING*.

WRITE <fs\_int>.

2.

DATA: gr\_data   TYPE REF TO DATA,

      gv\_int\_a  TYPE i,

      gv\_int\_b  TYPE i.

FIELD-SYMBOLS: <gfs\_int\_b> TYPE ANY.

gv\_int\_a = 5.

GET REFERENCE OF gv\_int\_a INTO gr\_data.

ASSIGN gr\_data->\* TO <gfs\_int\_b> *CASTING TYPE i*.

gv\_int\_b = <gfs\_int\_b>.

WRITE: gv\_int\_b.

**#**

[Содержание](#Содержание)

field-symbols на переменные можно объявлять непосредственно в месте их использования

DATA: lt\_tab TYPE STANDARD TABLE OF i WITH DEFAULT KEY.

lt\_tab = *VALUE #*( ( 1 ) ( 2 ) ( 3 ) ).

READ TABLE lt\_tab INDEX 1 ASSIGNING FIELD-SYMBOL(<ls\_one>).

WRITE <ls\_one>.

LOOP AT lt\_tab FROM 2 ASSIGNING FIELD-SYMBOL(<ls\_two>).

  WRITE <ls\_two>.

ENDLOOP.

## **Ссылочные переменные**

[Содержание](#Содержание)

DATA: gr\_data TYPE *REF TO* data.

Единственным обобщенным типом для ссылочных переменных может быть тип – ***data***.

**Указатель на перемнную**

FORM some\_procedure.

  DATA: lv\_counter TYPE i VALUE 5.

*GET REFERENCE* OF lv\_counter INTO gr\_dref. *" Получим указатель на локальную переменную*

ENDFORM.

В версии ABAP 7.40 появился удобный оператор ***REF #***для получения ссылочных переменных вместо GET REFERENCE

DATA: lt\_tab TYPE STANDARD TABLE OF i WITH DEFAULT KEY.

lt\_tab = VALUE #( ( 1 ) ( 2 ) ( 3 ) ).

DATA(lr\_tab) = *REF #*( lt\_tab ).

***Проверка присвоения***

Что касается проверки присвоения, то её необходимо выполнять с помощью ключевого слова ***IS BOUND***. IS INITIAL в данном контексте может привести к неправильной оценке.

DATA: gr\_dref TYPE REF TO data.

FIELD-SYMBOLS: <lfs\_counter> TYPE any.

START-OF-SELECTION.

  IF gr\_dref *IS INITIAL*. WRITE: / 'Ссылка пустая'.

  ENDIF.

  PERFORM some\_procedure.

    IF gr\_dref IS INITIAL. WRITE: / 'Ссылка пустая'.

   ELSE. WRITE: / 'Ссылка не пустая'.

    IF gr\_dref *IS BOUND*. WRITE: / 'Объект для ссылки существует'.

  ELSE. WRITE: / 'Объект для ссылки не существует'.

   ENDIF.

 ENDIF.

***Процедура de-referencing /разыменования/***

[Содержание](#Содержание)

DATA: gr\_data  TYPE REF TO string,

      gv\_data  TYPE string VALUE 'TEST'.

GET REFERENCE OF gv\_data INTO gr\_data.

*gr\_data->\** = 'NEW VALUE'.

WRITE *gr\_data->\**.

В том случае, *когда ссылочная переменная ссылается на структуру*, доступ к её компонентам осуществляется через ***->ИмяКомпонента***

TYPES: BEGIN OF ty\_struct,

    char1,

     char2,

  END OF ty\_struct.

DATA: gs\_struct TYPE ty\_struct.

DATA: gr\_struct TYPE REF TO ty\_struct.

gs\_struct-char1 = 'A'.

gs\_struct-char2 = 'B'.

GET REFERENCE OF gs\_struct INTO gr\_struct.

WRITE: / *gr\_struct->*char1, *gr\_struct->*char2.

**CREATE DATA**

[Содержание](#Содержание)

С помощью оператора CREATE DATA мы *можем создать анонимный объект в памяти*, привязав его к ссылочной переменной - время жизни такого объекта равносильно времени жизни ссылочной переменной

DATA: lr\_string TYPE REF TO string.

*CREATE DATA* lr\_string.

lr\_string->\* = 'Test'.

При этом есть возможность *указать тип динамически*

DATA: lr\_int TYPE REF TO DATA,

  lv\_type TYPE string VALUE 'i'.

FIELD-SYMBOLS: <lv\_int> TYPE any.

*CREATE DATA* lr\_int *TYPE (lv\_type)*.

ASSIGN lr\_int->\* TO <lv\_int>.

<lv\_int> = 10.

WRITE <lv\_int>.

## **Runtime type services**

[Содержание](#Содержание)

*Для получения описания типа данных, созданного at run-time* можно воспользоваться инструментом RTTS. По сути это определенная иерархия классов, вызывая методы которых можно получить полное описание типа данных. Кроме описательной возможности, данный инструмент позволяет динамически создавать типы данных at run-time.

DATA:  lr\_ref  TYPE REF TO DATA,

  lo\_type TYPE REF TO cl\_abap\_elemdescr.

*\* Получаем описание типа С длинной 40 символов*

lo\_type = cl\_abap\_elemdescr=>get\_c( 40 ).

*\* Создаем объект данных по описательному объекту*

CREATE DATA lr\_ref *TYPE* *HANDLE* lo\_type.

***Работа с датой***

ABAP stores dates in the format *YYYYMMDD*. You can access individual components of the date using *offset* and *length*.

DATA: Year(4) TYPE C, Month(2) TYPE C, Day(2) TYPE C.

Year = var\_date(4).

Month = var\_date+4(2).

Day = var\_date +6(2).

As well as the type *D*, ABAP has a built-in data type utclong, which represents a ***timestamp***. This data type has an accuracy of 100 nanoseconds and follows the format *YYYY-MM-DDTHH:MM:SS.sssssssZ*, where T is the delimiter between date and time, and Z denotes the time zone Zulu, which stands for UTC.

Даты передается и возвращается в стандартном формате типа SY-DATUM (YYYYMMDD, YYYY - Год, MM - Месяц, DD - День):

data: begda like sy-datum,  
        endda like sy-datum.

# Некоторые полезные ФМ и Программы в SAP BW

<https://helpbw.wordpress.com/2017/04/19/%D0%BD%D0%B5%D0%BA%D0%BE%D1%82%D0%BE%D1%80%D1%8B%D0%B5-%D0%BF%D0%BE%D0%BB%D0%B5%D0%B7%D0%BD%D1%8B%D0%B5-%D1%84%D0%BC-%D0%B2-sap-bw/>

# Function Module related on Date calculations

<https://help.sap.com/docs/SUPPORT_CONTENT/abap/3353525682.html>

# Работа с датами в ABAP. Полезные ФМ SAP.

<https://dzen.ru/a/Ylm5OAQvVnpUpsFK>

### ***SAP ABAP функции для работы с датой и временем***

<https://razumsap.blogspot.com/2014/03/sap-abap.html>

This code will give you the date in the long text format like 'December 02, 2011'. You may change the code accordingly to print the date with long MONTH name.

DATA: LONG\_DATE(20).

PERFORM GET\_LONG\_DATE USING LONG\_DATE.

WRITE: LONG\_DATE.

FORM GET\_LONG\_DATE USING DATE.

DATA: T\_MONTH\_NAMES LIKE TABLE OF T247 WITH HEADER LINE.

CALL FUNCTION 'MONTH\_NAMES\_GET'

EXPORTING

LANGUAGE = SY-LANGU

TABLES

MONTH\_NAMES = T\_MONTH\_NAMES

.

DATA: YEAR(4) TYPE C,

MONTH(2) TYPE C,

DAY(2) TYPE C.

YEAR = SY-DATUM+(4).

MONTH = SY-DATUM+4(2).

DAY = SY-DATUM+6(2).

Names like "NAME()" or "NAME+", as in "SY-DATUM+(4)", are identical to "NAME". These are no longer allowed in Release 4.0.

READ TABLE T\_MONTH\_NAMES INDEX ( MONTH ).

CONCATENATE T\_MONTH\_NAMES-LTX ' ' DAY INTO DATE SEPARATED BY SPACE.

CONCATENATE DATE ',' INTO DATE.

CONCATENATE DATE YEAR INTO DATE SEPARATED BY SPACE.

WRITE / DATE.

ENDFORM.

Часто встречается потребность модификации даты на определённый промежуток времени: прибавить год, отнять месяц или заданное количество дней. Последнее можно выполнить простой арифметической операцией вида

DATA: lv\_data TYPE d TYPE sy-datum.

lv\_date = lv\_date + 100. " Прибавить 100 дней к текущей дате:

Что же касается *месяца* или *года*, можно воспользоваться стандартным ФМ-ом вида

DATA: dt TYPE sy-datum.

CALL FUNCTION '*RP\_CALC\_DATE\_IN\_INTERVAL*'

  EXPORTING

    date      = sy-datum                  " Дата с которой работаем

    days      = 0                         " 0 Дней

    months    = 0                         " 0 Месяцев

    signum    = '-'                       " Операция вычитания

    years     = 1                         " 1 Год

  IMPORTING

    calc\_date = dt.                       " Результат

Еще одним из стандартных вариантов - использовать класс**CL\_RECA\_DATE** и его метода (ADD\_TO\_DATE), если у вас присутствует RE модуль в системе.