LOGALI

TEORÍA
Sentencias en ABAP

SAP ABAP Programación

Contenido

1. Cadenas de caracteres

1.1. Conceptos

Se entiende por cadenas de caracteres cualquier objeto de los tipos: **c**, **d**, **n, t** y **string**. Si los componentes de una estructura son todos del tipo carácter podemos operar con la estructura como si fuera una única cadena de caracteres.

1.2. Encadenando cadenas de caracteres

Usamos la instrucción **CONCATENATE**, por ejemplo:

```
CONCATENATE s1 s2 INTO s3.
```

Si queremos que las dos cadenas estén separadas por un espacio en blanco ponemos:

```
CONCATENATE s1 s2 INTO s3 SEPARATED BY SPACE.
```

Es conveniente declarar s3 del tipo **string** para no correr el riesgo de quedarse sin espacio al encadenar varios objetos.

```
DATA: s1(20)  TYPE  c VALUE 'hola',
      s2(20) TYPE c VALUE 'amigos',
      s3       TYPE string.
```

1.3. Separando cadenas de caracteres

Para separar cadenas usamos la instrucción SPLIT

```
SPLIT s3 AT SPACE INTO s1 s2.
```

Podemos separar una cadena de caracteres en una tabla. En este caso cada palabra aparecerá en una línea distinta de la tabla:

```
SPLIT s3 AT SPACE INTO TABLE t.
```

En este caso declaramos t como una tabla del tipo string:

```
DATA t TYPE TABLE OF string.
```

1.4. Buscando caracteres

Para buscar un patrón de caracteres (por ejemplo s1) en una cadena de caracteres (por ejemplo s2) usamos la instrucción **SEARCH**

```
DATA: s1 TYPE string VALUE 'amigos',
      s2 TYPE string VALUE 'hola amigos'.

SEARCH s2 FOR s1.
```

Si el sistema encuentra el patrón s1 en s2 la variable de sistema sy-subrc guarda un **5** (recuerde que el sistema empieza a contar desde cero). La variable de sistema sy-fdpos guarda la posición en que comienza el patrón s1 dentro de s2.

Podemos buscar una palabra en una cadena de caracteres, por ejemplo:

```
SEARCH s2 FOR 'amigos'.
```

Podemos buscar una cadena que comience por un patrón, por ejemplo:

```
SEARCH s2 FOR 'ami*'.
```

Podemos buscar una cadena que finalice por un patrón, por ejemplo:

```
SEARCH s2 FOR '*gos'.
```

1.5. Reemplazando caracteres

Para reemplazar una cadena de caracteres usamos la instrucción REPLACE. El sistema busca el patrón s1 en s3 y si lo encuentra lo sustituye por s2.

```
DATA: s1 TYPE string VALUE 'hola',
      s2 TYPE string VALUE 'adiós',
      s3 TYPE string VALUE 'hola amigos'.

REPLACE s1 WITH s2 INTO s3.
```

2. Bifurcaciones condicionales

En ABAP existen varias formas de ejecutar distintas secuencias de sentencias, según determinadas condiciones. A continuación, se detallan todas las posibilidades que nos ofrece ABAP.

2.1. Sentencia IF/ELSE/ENDIF

En la **construcción IF** puede definir **cualquier expresión lógica** como condición de verificación. Si se cumple la condición, el sistema ejecuta el bloque de sentencias correspondiente. De lo contrario, se verifica la condición específica en la siguiente bifurcación **ELSEIF** (es posible incluir varias bifurcaciones). Si no cumple ninguna de las condiciones específicas, se ejecuta la bifurcación **ELSE**, si es que existe. Las bifurcaciones **ELSEIF** y **ELSE** son opcionales.

```
IF gv_var > 0.
* Sentencias
ELSEIF gv_var = 0.
* Sentencias
ELSE.
* Sentencias
ENDIF.
```

3. Modularización

3.1. Introducción

Una **unidad de modularización** es una parte de un programa en la que se encapsula una función concreta. Una parte del código fuente se almacena

en un módulo para mejorar la transparencia del programa, para poder usar la función correspondiente del programa varias veces sin tener que volver a implementar el código fuente entero en cada ocasión (consulte el gráfico anterior).

El resultado obtenido cuando un programa pasa a estar más orientado a las funciones es la mejora de la transparencia: el programa divide la tarea global en subfunciones, que son responsabilidad de las unidades de modularización correspondientes.

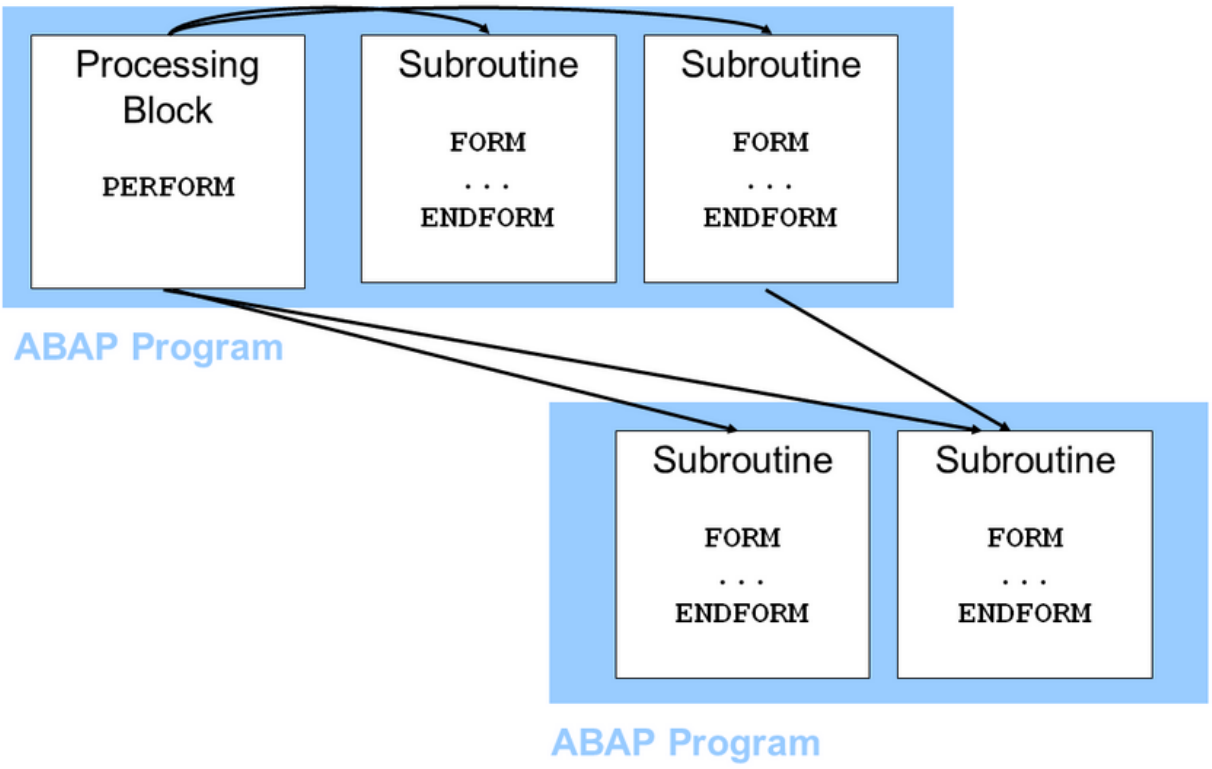
La modularización facilita la actualización de programas, ya que permite que las modificaciones se realicen únicamente en la función, o las correcciones en las unidades de modularización respectivas, y no en varios puntos del programa principal. Asimismo, también le permite procesar una llamada “como unidad” en el debugger durante la ejecución del programa y visualizar posteriormente el resultado. De este modo, resulta mucho más fácil encontrar el origen del error.

3.2. Modularización de programa local

Existen dos técnicas para la modularización de programa local en el lenguaje de

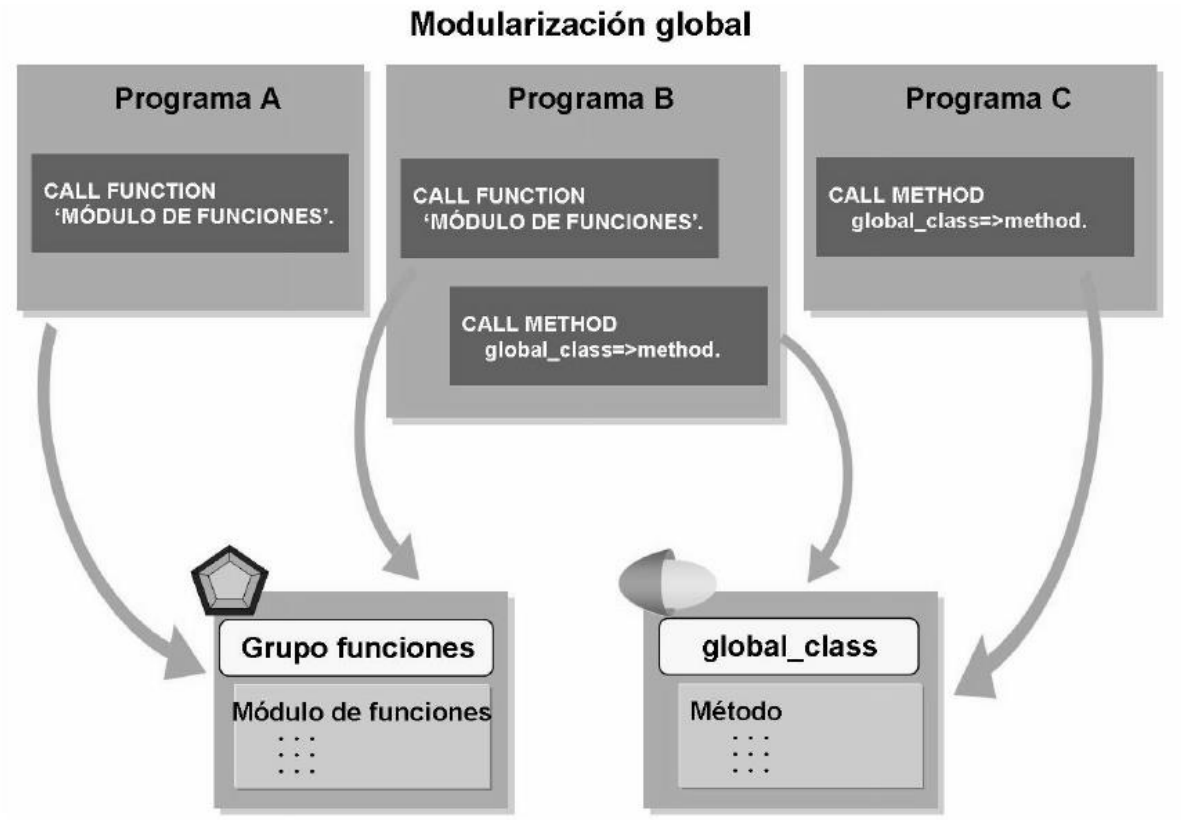
programación ABAP:

- **Subrutinas**, también conocidas como rutinas **FORM**
- **Métodos** en clases locales



Con ambas técnicas de modularización local, las unidades de modularización sólo están disponibles en el programa en el que implementan. Para llamar el módulo local, no es necesario cargar ningún otro programa en el contexto de usuario en tiempo de ejecución. Las clases locales, los métodos y las subrutinas pueden tener el mismo nombre en diferentes programas sin causar conflictos. Esto se debe a que el código fuente de los programas se trata por separado en la memoria principal del servidor de aplicación.

3.3. Modularización global



También existen dos técnicas para la modularización global en el lenguaje de programación ABAP:

- Módulos de funciones que están organizados en grupos de funciones
- Métodos en clases globales

Cualquier número indefinido de programas puede utilizar simultáneamente las unidades de modularización definidas globalmente. Las unidades de modularización definidas globalmente se almacenan centralmente en el Repository y se cargan cuando son necesarias (es decir, cuando se llaman) el contexto del programa de llamada.