Texto estructurado (ST)

El texto estructurado se compone de una serie de instrucciones que se pueden ejecutar, como sucede con los lenguajes superiores, de forma condicionada ("IF..THEN..ELSE") o en bucles secuenciales (WHILE..DO). Ejemplo:

```
IF value < 7 THEN
    WHILE value < 8 DO
        value := value + 1;
    END_WHILE;
END_IF;
```

Expresiones

Una expresión es una construcción que devuelve un valor después de su evaluación. Las expresiones se componen de operadores y operandos. Un operando puede ser una constante, una variable, una llamada a funciones u otra expresión.

Evaluación de expresiones

La evaluación de una expresión se realiza mediante la ejecución de los operadores según determinadas reglas de enlace. El operador con el enlace más fuerte se ejecuta primero, después el operador que le sigue en intensidad de enlace, etc., hasta que se hayan ejecutado todos los operadores. Los operadores con la misma fuerza de enlace comienzan a ejecutarse desde la izquierda hacia la derecha.

A continuación se muestra una tabla con los operadores ST por orden de fuerza de enlace.

Operación	Símbolo	Fuerza de enlace
Entre paréntesis	(expresión)	Enlace más fuerte
Llamada a funciones	Nombre de función (lista de parámetros)	
Potenciar	**	
Negar	-	
Formación de complemento	NOT	
Multiplicar	*	
Dividir	/	
Módulo	MOD	
Sumar	+	
Restar	-	
Comparar	<,>,<=,>=	
Igualdad	=	
Desigualdad	<>	
Bool AND	AND	
Bool XOR	XOR	
Bool OR	OR	Enlace más débil

A continuación se indican las siguientes instrucciones en ST ordenadas en forma de tabla y con un ejemplo:

Tipo de instrucción	Ejemplo
Asignación	A:=B; CV := CV + 1; C:=SIN(X);
Llamada a un Bloque de Funciones y uso de la salida FB	CMD_TMR(IN := %IX5, PT := 300);A:=CMD_TMR.Q;
RETURN	RETURN;
IF	IF D:=B*B;IF D<0.0 THEN C:=A;ELSIF D=0.0 THEN C:=B;ELSE C:=D;END_IF;
CASE	CASE INT1 OF1: BOOL1 := TRUE;2: BOOL2 := TRUE;ELSE BOOL1 := FALSE; BOOL2 := FALSE;END_CASE;

FOR	FOR J:=101;FOR I:=1 TO 100 BY 2 DO IF ARR[I] = 70 THEN J:=I; EXIT; END_IF;END_FOR;
WHILE	WHILE J<= 100 AND ARR[J] <> 70 DO J:=J+2;END_WHILE;
REPEAT	REPEAT J:=J+2;UNTIL J= 101 OR ARR[J] = 70END_REPEAT;
EXIT	EXIT;
Instrucción vacía	;

Instrucciones en el lenguaje de texto estructurado

Tal como el nombre lo indica, el texto estructurado está concebido para la programación estructurada; es decir, que para determinadas construcciones de uso frecuente, tales como bucles secuenciales, el lenguaje ST ofrece estructuras establecidas para la programación. Esto proporciona la ventaja de reducir la probabilidad de errores y conferir mayor claridad al programa. Comparemos, por ejemplo, dos secuencias de programa con idéntico significado en los lenguajes IL y ST:

Un bucle secuencial para el cálculo de segundas potencias en lenguaje IL:
bucle secuencial:

```
LD Zaehler
EQ 0
JMPC ende
LD Var1
MUL 2
ST Var1
LD Zaehler
SUB 1
ST Zaehler
JMP schleife
ende:
LD Var1
ST Erg
```

El mismo bucle secuencial programado en lenguaje ST se presentaría como sigue:

```
WHILE Zaehler<>0 DO
    Var1:=Var1*2;
    Zaehler:=Zaehler-1;
END_WHILE

Erg:=Var1;
```

Como se puede observar, la programación en el lenguaje ST del bucle secuencial no sólo es más breve, sino también considerablemente más fácil de leer, sobre todo si nos imaginamos bucles secuenciales entrelazados en construcciones de mayor envergadura.
Las distintas estructuras en lenguaje ST tienen el siguiente significado:

Operador de asignación

En el lado izquierdo de una asignación se encuentra un operando (variable, dirección) al cual se asigna el valor de la expresión del lado derecho junto con el operador de asignación :=
Ejemplo:
Var1 := Var2 * 10;
Después de la ejecución de esta línea, Var1 tiene un valor diez veces superior a Var2.

Llamada a Bloques de Funciones en el lenguaje ST

Una llamada a un Bloque de Funciones en lenguaje ST se realiza escribiendo el nombre de la instancia del Bloque de Funciones y asignando a continuación, y entre paréntesis, los valores que desee de los parámetros. En el siguiente ejemplo se llama a un temporizador con asignaciones para los parámetros IN y PT. A continuación, se asigna la variable de resultado Q a la variable A.
La variable de resultado se activa, como en el lenguaje IL, con el nombre del Bloque de Funciones seguido de un punto y del nombre de la variable:

```
CMD_TMR(IN := %IX5, PT := 300);
A:=CMD_TMR.Q
```

Instrucción RETURN

La instrucción RETURN se puede utilizar para terminar una función que, por ejemplo, es dependiente de una condición.

Instrucción IF

Con la instrucción IF se puede comprobar una condición y ejecutar instrucciones en función de esta condición.
Sintaxis:

```
IF <Boolscher_Ausdruck1> THEN
<IF_Anweisungen>
{ELSIF <Boolscher_Ausdruck2> THEN
<ELSIF_Anweisungen1>
.
.
ELSIF <Boolscher_Ausdruck n> THEN
<ELSIF_Anweisungen n-1>
ELSE
<ELSE_Anweisungen>}
END_IF;
```

La parte entre llaves {} es opcional.
Si el valor de < Boolscher_Ausdruck1> es TRUE, sólo se ejecuta <IF_Anweisungen>, ninguna de las demás instrucciones.
De lo contrario, las expresiones de Bool, empezando por < Boolscher_Ausdruck2>, se evalúan sucesivamente hasta que una de las expresiones tenga el valor TRUE. De ese modo, sólo se evalúan las instrucciones que se encuentran detrás de esta expresión de Bool y delante del siguiente ELSE o ELSIF. Si ninguna de las expresiones de Bool tiene el valor TRUE, se evalúa únicamente <ELSE_Anweisungen>.

Ejemplo:

```
IF temp<17
THEN heizung_an := TRUE;
ELSE heizung_an := FALSE;
END_IF;
```

En este caso, la calefacción se enciende cuando la temperatura baja de los 17 grados; de lo contrario permanece apagada.

Instrucción CASE

Con la instrucción CASE se pueden reunir varias instrucciones condicionadas con la misma variable de condición en una sola construcción.
Sintaxis:

```
CASE <Var1> OF
<Wert 1>: <Anweisung 1>
<Wert 2>: <Anweisung 2>
...
<Wert n>: <Anweisung n>
ELSE <ELSE-Anweisung>
END_CASE;
```

Una instrucción CASE se ejecuta según el siguiente esquema:

- Si la variable en <Var1> tiene el valor <Wert i>, se ejecuta la instrucción <Anweisung i>.
- Si <Var 1> no tiene ninguno de los valores indicados, se ejecuta <ELSE-Anweisung>.

- Si ha de ejecutarse la misma instrucción para varios valores de la misma variable, estos valores se pueden escribir uno tras otro, separados por comas, condicionando así la instrucción común.

Ejemplo:

```
CASE INT1 OF
1, 5: BOOL1 := TRUE;
BOOL3 := FALSE;
2: BOOL2 := FALSE;
BOOL3 := TRUE;
ELSE
BOOL1 := NOT BOOL1;
BOOL2 := BOOL1 OR BOOL2;
END_CASE;
```

Bucle secuencial FOR

Con el bucle secuencial FOR se pueden programar procesos repetidos.
Sintaxis:

```
INT_Var :INT;

FOR <INT_Var> := <INIT_WERT> TO <END_WERT> {BY <Schrittgröße>} DO
<Anweisungen>
END_FOR;
```

La parte entre llaves {} es opcional.
<Anweisungen> se ejecutará mientras el contador <INT_Var> no sea superior a <END_WERT>. Esto se comprueba antes de la ejecución de <Anweisungen>, de modo que <Anweisungen> no se ejecuta nunca si <INIT_WERT> es superior a <END_WERT>.
Siempre que se haya ejecutado <Anweisungen>, <INT_Var> aumenta en <Schrittgröße>. El tamaño de paso puede tener cualquier valor entero. Si falta éste, se ajusta a 1. Por lo tanto, el bucle secuencial se tiene que terminar, ya que <INT_Var> sólo se hace más grande.
Ejemplo:

```
FOR Zaehler:=1 TO 5 BY 1 DO
Var1:=Var1*2;
END_FOR;
Erg:=Var1;
```

Supongamos que la variable Var1 se haya establecido por defecto con el valor 1; en este caso, después del bucle secuencial FOR, tendrá el valor 32.

Bucle secuencial WHILE

El bucle secuencial WHILE se puede utilizar igual que el bucle secuencial FOR, con la diferencia de que la condición de cancelación puede ser cualquier expresión de Bool. Esto significa que se indica una condición que, en caso de cumplirse, produce la ejecución del bucle secuencial.
Sintaxis:

```
WHILE <Boolscher Ausdruck> DO
<Anweisungen>
END_WHILE;
```

La ejecución de <Anweisungen> se repite mientras <Boolscher Ausdruck> sea TRUE. Si <Boolscher Ausdruck> tiene el valor FALSE desde la primera evaluación, <Anweisungen> no se ejecuta nunca. Si <Boolscher Ausdruck> no tiene nunca el valor FALSE, <Anweisungen> se repite de forma infinita, con lo cual se produce un error de tiempo de funcionamiento.

Nota: el programador ha de asegurarse de que no se produzca ningún bucle secuencial infinito y, para ello, tendrá que modificar la condición en la parte de instrucción del bucle secuencial, por ejemplo, aumentando o reduciendo el conteo de un contador.

Ejemplo:

```
WHILE Zaehler<>0 DO
    Var1 := Var1*2;
    Zaehler := Zaehler-1;
END_WHILE
```

Los bucles secuenciales WHILE y REPEAT son, en cierto sentido, más potentes que el bucle secuencial FOR, ya que no es necesario saber antes de la ejecución el número de pasos del bucle secuencial. En ciertos casos, sólo podrá trabajar con estos dos tipos de bucles secuenciales. No obstante, si el número de pasos de bucle secuencial está claro, se dará preferencia a un bucle secuencial FOR, ya que no se permiten los bucles secuenciales infinitos.

Bucle secuencial REPEAT

El bucle secuencial REPEAT se distingue de los bucles secuenciales WHILE por el hecho de que la condición de cancelación se comprueba solamente después de la ejecución del bucle secuencial, de modo que el bucle secuencial se ejecuta por lo menos una vez, independientemente de la condición de cancelación.
Sintaxis:

```
REPEAT
<Anweisungen>
UNTIL <Boolscher Ausdruck>
END_REPEAT;
```

<Anweisungen> se ejecuta hasta que <Boolscher Ausdruck> es TRUE.
Si <Boolscher Ausdruck> es TRUE desde la primera evaluación, <Anweisungen> se ejecuta exactamente una vez. Si a <Boolscher Ausdruck> no se le asigna nunca el valor TRUE, <Anweisungen> se repite de forma infinita, con lo cual se produce un error de tiempo de funcionamiento.

Nota: el programador ha de asegurarse de que no se produzca ningún bucle secuencial infinito y, para ello, tendrá que modificar la condición en la parte de instrucción del bucle secuencial, por ejemplo, aumentando o reduciendo el conteo de un contador.

Ejemplo:

```
REPEAT
    Var1 := Var1*2;
    Zaehler := Zaehler-1;
UNTIL
    Zaehler=0
END_REPEAT
```

Instrucción EXIT

Si la instrucción EXIT se encuentra en un bucle secuencial FOR, WHILE o REPEAT, el bucle secuencial interior se termina, independientemente de la condición de cancelación.