13. Einführung in die Programmiersprache Strukturierter Text (ST)

13.1 Übersicht

Strukturierter Text (ST, auch SCL) ist eine der sechs in IEC 61131-3 festgeschriebenen Programmiersprachen für Automatisierungstechnik. Sie orientiert sich an PASCAL und enthält sowohl Sprachelemente dieser Sprache als auch SPS-typische Elemente.

Besonders geeignet ist ST für alle Aufgaben, die sich mit mathematischen Formeln beschreiben lassen, wie die **Programmierung komplexer Algorithmen, mathematischer Funktionen und für Rezept- und Datenverwaltung**. Solche Programmteile werden mit ST bedeutend vereinfacht.

Typisch für Strukturierten Text sind Anweisungen, die wie in Hochsprachen bedingt (IF..THEN..ELSE) oder in Schleifen (WHILE..DO) ausgeführt werden können. Für SPS-typische Aufgaben wie Timer, Trigger, Counter und RS-FlipFlop kommen auch in ST die Funktionsbausteine der Standardbibliothek zum Einsatz. Regeln und Syntax des Aufrufs solcher Bausteine sowie der Aufruf von Funktionen und Programmen werden in Abschnitt 13.3 behandelt.

Im System Simatic S7 / Step7 ist diese Sprache als Softwaretool **S7-SCL normkonform zu IEC 61131** verfügbar.

Bausteine in ST können mit Bausteinen kombiniert werden, die in anderen Sprachen entwickelt wurden. Das gilt auch für S7-SCL.

13.2 Anweisungen, Ausdrücke und ihre Auswertung

ST arbeit mit **Anweisungen** und Ausdrücken und hält spezielle Operatoren für die Auswertung von Ausdrücken bereit. Ein Ausdruck ist ein Konstrukt, das nach seiner Auswertung einen Wert zurückliefert.

Ausdrücke sind zusammengesetzt aus Operatoren und Operanden. Ein Operand kann eine Konstante, eine Variable, ein Funktionsaufruf oder ein weiterer Ausdruck sein.

Eine besondere Bedeutung erhält der **Zuweisungsoperator := .** Hierbei steht auf der linken Seite einer Zuweisung ein Operand (Variable, Adresse), dem der Wert des Ausdrucks auf der rechten Seite mit dem Operator := zugewiesen wird.

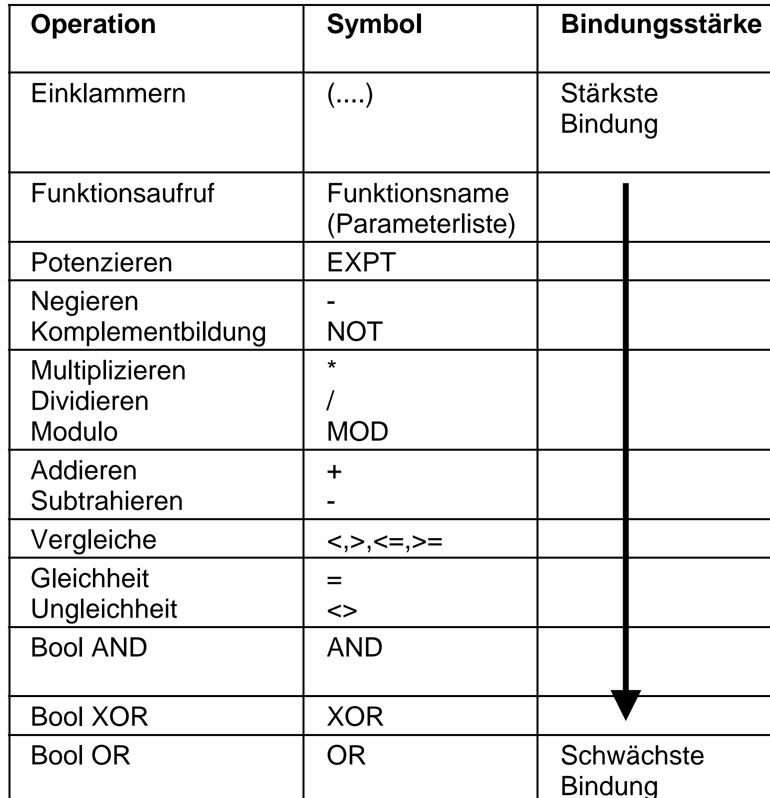
Alle Anweisungen sind mit dem **Zeichen “ ; “** abzuschließen.

Beispiele für die Schreibweise von Anweisungen:

```
Var_1 := Var_1+ 100;

OUT_1 := Var_1 XOR Var_2;

Fehlerbit:= Ergebnis>1200 ;

Neuwert := Altwert * 10 ;

Instanz_FB1(IN:=Var_1,Par:=10);
```

Nachfolgende Tabelle enthält alle Operatoren für ST (Quelle: Online-Hilfe von CoDeSys 2.3.7.3).
Wie bei mathematischen Formeln ist die Bindungsstärke (der „Vorrang“) der Operationen festgelegt.

Die speziellen Anweisungen in ST zeigt tabellarisch geordnet die zweite Tabelle jeweils zusammen mit einem Beispiel zur Darstellung der Syntax (Quelle: Online-Hilfe von CoDeSys 2.3.7.3).



Operation	**Symbol**	**Bindungsstärke**
Einklammern	(....)	Stärkste Bindung
Funktionsaufruf	Funktionsname (Parameterliste)	
Potenzieren	EXPT	
Negieren Komplementbildung	- NOT	
Multiplizieren Dividieren Modulo	* / MOD	
Addieren Subtrahieren	+ -	
Vergleiche	<,>,<=,>=	
Gleichheit Ungleichheit	= <>	
Bool AND	AND	
Bool XOR	XOR	
Bool OR	OR	Schwächste Bindung

Anweisungsart	**Beispiel**
Zuweisung	A:=B; CV := CV + 1; C:=SIN(X);
Aufruf eines Funktionsblocks und Benutzung der FB-Ausgabe	CMD_TMR(IN := %IX5, PT := 300); A:=CMD_TMR.Q
RETURN	RETURN;

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 OF 1: BOOL1 := TRUE; 2: BOOL2 := TRUE; ELSE BOOL1 := FALSE; BOOL2 := FALSE; END_CASE;

FOR	J:=101;
	FOR I:=1 TO 100 BY 2 DO
	IF ARR[I] = 70 THEN
	J:=I;
	EXIT;
	END_IF;
	END_FOR;

WHILE	J:=1; WHILE J<= 100 AND ARR[J] <> 70 DO J:=J+2; END_WHILE;
REPEAT	J:=-1; REPEAT J:=J+2; UNTIL J= 101 OR ARR[J] = 70 END_REPEAT;
EXIT	EXIT;
Leere Anweisung	;

- **Die Anwendung der Anweisungen im einzelnen:**

<table>
<tr><td>IF –
Anweisung</td><td>Mit dieser Anweisung kann man Bedingungen abprüfen und davon abhängig Anweisungen ausführen.

IF Boolscher Ausdruck1 THEN
    IF Anweisungen
ELSIF Boolscher Ausdruck2 THEN
    ELSIF Anweisungen
ELSIF.....
:
ELSE
    ELSE Anweisungen
END_IF;

• Die Anweisungen werden der Reihe nach ausgewertet.
• Ist der Boolsche Ausdruck1 TRUE, werden nur IF Anweisungen ausgewertet.
• Andernfalls werden die anderen Boolschen Ausdrücke geprüft und die zum ersten TRUE gehörenden ELSIF Anweisungen ausgewertet.
• Ist kein Boolscher Ausdruck TRUE, werden nur die ELSE Anweisungen ausgewertet.
• Die Programmteile ELSIF und ELSE sind optional.</td></tr>
<tr><td>CASE -
Anweisung</td><td>Mit dieser Anweisung kann man mehrere bedingte Anweisungen zusammenfassen.

CASE Variable1 OF
Wert1: Anweisung1
Wert2: Anweisung2
:
Wertn: Anweisungen
ELSE ELSE Anweisungen
END_CASE;

• Es werden die Anweisungen bearbeitet, deren Wert-Zuordnung dem der Variablen entspricht.
• Trifft kein Wert zu, wird die ELSE Anweisung bearbeitet.
• Treffen mehrere Werte zu, werden die zugehörigen Anweisungen bearbeitet. Diese können durch Komma getrennt gemeinsam hintereinander geschrieben werden.
• Der Programmteil ELSE ist optional.
• Wertebereiche werden mit zwei Punkten geschrieben: z.B. 2..10</td></tr>
</table>

<table>
<tr><td>FOR – Schleife</td><td>Mit FOR-Schleifen kann man wiederholte Vorgänge programmieren.

Variable1:INT;
FOR Variable1:=Initialwert TO Endwert BY Schrittgrösse DO
        Anweisungen
END_FOR;

• Die Anweisungen werden solange bearbeitet, solange die Variable1 nicht den Endwert überschreitet.
• Immer wenn die Anweisungen bearbeitet wurden, wird Variable1 um die Schrittgröße erhöht.
• Die Angabe einer Schrittgröße ist optional. Wird keine Schrittgröße angegeben, so wird sie mit Schrittgröße 1 ausgeführt.
• Vorsicht vor Endlosschleifen! Der Endwert darf z.B. nicht ein Grenzwert der Variable1 sein!</td></tr>
<tr><td>WHILE – Schleife</td><td>WHILE-Schleifen werden benutzt wie FOR-Schleifen mit dem Unterschied, dass man mit einer Boolschen Bedingung die Ausführung bestimmen und auch abbrechen kann.

WHILE Variable_Bool DO
        Anweisungen
END_WHILE;

• Die Anweisungen werden solange wiederholt bearbeitet, solange die Variable_Bool den Wert TRUE hat.
• Ist der Wert zu Beginn FALSE, werden die Anweisungen nicht bearbeitet.
• Vorsicht vor Endlosschleifen! Wenn die Variable_Bool niemals FALSE wird, entstehen Endlosenschleifen. Der Programmierer muss dies verhindern, z.B. durch Programmieren eines Zählers.
Ist die Anzahl der Schleifendurchläufe bekannt, sind FOR-Schleifen zu bevorzugen!</td></tr>
<tr><td>REPEAT–Schleife</td><td>REPEAT-Schleifen unterscheiden sich von WHILE-Schleifen nur dadurch, dass die Abbruchbedingung erst nach dem Ausführen der Schleife geprüft wird. Dadurch wird die Schleife stets mindestens einmal durchlaufen.

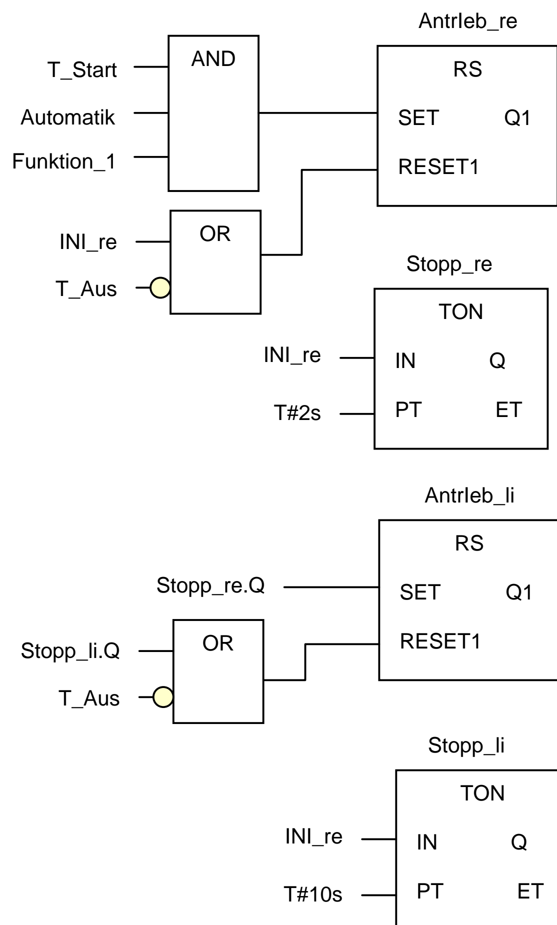
REPEAT
        Anweisungen
UNTIL Variable_Bool
END_REPEAT;

• Die Anweisungen werden solange bearbeitet, bis die Variable_Bool den Wert TRUE hat.
• Hat die Variable_Bool bereits bei der ersten Auswertung den Wert TRUE, werden die Anweisungen genau einmal ausgeführt.
• Vorsicht vor Endlosschleifen! Hat die Variable_Bool niemals den Wert TRUE, entstehen Endlosschleifen. Der Programmierer muss dies verhindern, z.B. durch Programmieren eines Zählers.
Ist die Anzahl der Schleifendurchläufe bekannt, sind FOR-Schleifen zu bevorzugen!</td></tr>
<tr><td>EXIT-Anweisung</td><td>Mit EXIT werden FOR-, WHILE- und REPEAT-Schleifen beendet ungeachtet anderer Abbruchbedingungen.</td></tr>
<tr><td>RETURN-Anweisung</td><td>Mit RETURN kann man einen Baustein verlassen.</td></tr>
</table>

- **Beispiel 1:** Nachfolgend angegebenes Detail einer Antriebssteuerung wird in ST überführt.

FUNKTION_BLOCK Antrieb

VAR
T_Start:BOOL;
T_Aus:BOOL;
Automatik:BOOL;
Funktion_1:BOOL;
INI_re:BOOL;
Antrieb_re: RS;
Antrieb_li: RS;
Stopp_re: TON;
Stopp_li: TON;
END_VAR



Programm in ST: (bei gleicher Variablendeklaration wie oben)

```
Antrieb_re(SET:=T_Start AND Automatik AND Funktion_1,RESET1:=INI_rechts OR NOT T_Aus);

Stopp_re(IN:=INI_re,PT:=T#2s);

Antrieb_li(SET:=Stopp_re.Q,RESET1:=Stopp_li.Q OR NOT T_Aus);

Stopp_li(IN:=INI_re,PT:=T#10s);
```

Der **Aufruf einer Instanz** des o.a. Funktionsblockes mit Namen *Antrieb_1* z.B. in der POE PLC_PRG lautet in der Sprache ST:

```
PROGRAM  PLC_PRG
VAR
Antrieb_1:Antrieb;
END_VAR

Antrieb_1;
```

oder bedingt mit Anweisung CASE, wobei Bedingung eine Variable vom Typ INT ist:

```
CASE Bedingung_1 OF
1:Antrieb_1;
END_CASE;
```

- **Beispiel 2:** Anwendung der Anweisung CASE

Die **bedingte Abarbeitung von Programmteilen** durch Anwendung der Anweisung CASE verdeutlicht nachfolgendes Programm. Es wurden drei gleichartige POE vom Typ FB gestaltet, die Takte von 0,5 Hz, 1 Hz und 2 Hz erzeugen. Ihre Namen sind *Takt05Hz, Takt1Hz* bzw. *Takt2Hz.*

Durch Vorgabe des Wertes 1 oder 2 einer Variablen *Bedingung:INT* wird nun im zyklischen Programm PLC_PRG jeweils eine Instanz dieser POE **bedingt** aufgerufen. Dadurch arbeitet entweder der Taktgeber 1Hz oder der Taktgeber 2Hz.

Bei allen anderen Werten der Variablen *Bedingung* arbeitet dagegen der Taktgeber 0,5 Hz.

Bei der Simulation dieses Beispiels wird auch sofort die **Gefahr beim „Abschalten von Programmteilen“** deutlich: So können Taktsignale ihren Zustand TRUE behalten, auch wenn die POE nicht bearbeitet wird. Solche Zustände sind bei konkreten Applikationen stets zu prüfen und evtl. programmtechnisch zu unterbinden!

```
PROGRAM PLC_PRG
VAR
Bedingung: INT;
Instanz_Takt05Hz:Takt05Hz;
Instanz_Takt1Hz: Takt1Hz;
Instanz_Takt2Hz:Takt2Hz ;
END_VAR

CASE Bedingung OF
1:Instanz_Takt1Hz;
2:Instanz_Takt2Hz;
ELSE
Instanz_Takt05Hz;
END_CASE;
```

Bild 13-1 zeigt die Online-Sicht dieses Programmbeispiels, wobei die Bedingung den Wert 20 hat, also ungleich 1 oder 2 ist. Dementsprechend arbeitet nur der Taktgeber 0,5 Hz.

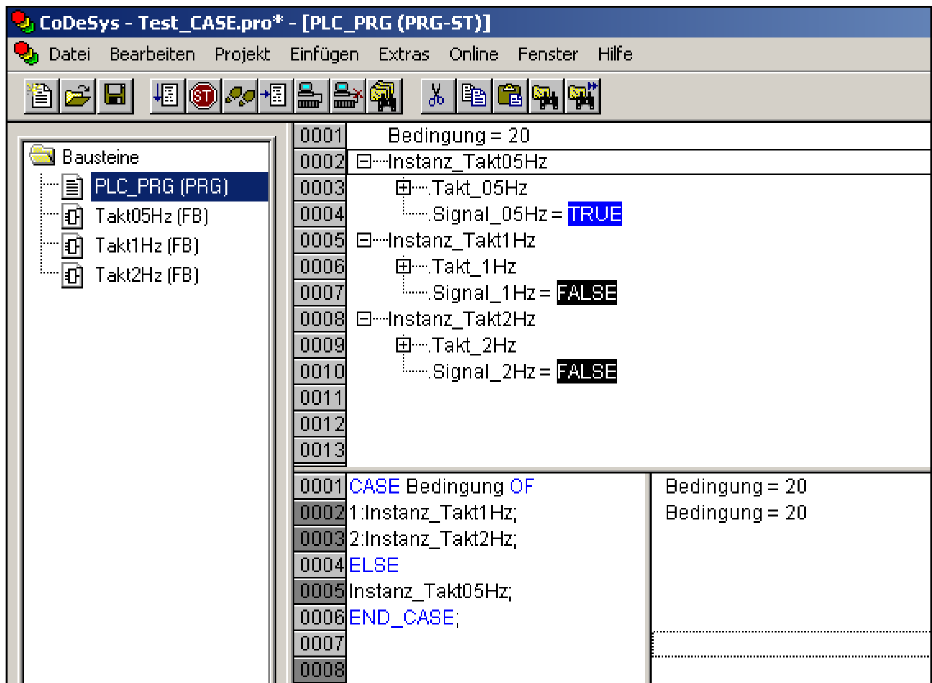


Bild 13-1: Online-Sicht der bedingten Programmbearbeitung mit CASE Anweisung

- **Beispiel 3**: Anwendung der Anweisung IF

Eine ähnliche Wirkung kann auch mit den Anweisungen IF / ELSEIF / ELSE erzielt werden. **Bild 13-2** zeigt das Programm für die gleiche Zielstellung wie Beispiel 2. Die Bedingungen mit Namen *Bedingung1* und *Bedingung2* sind hier Boolesche Variablen. Im Bild wurde Bedingung2 auf TRUE geschaltet. Folgerichtig arbeitet allein Taktgeber 2Hz. Hat keine der beiden Bedingungen den Wert TRUE, arbeitet Taktgeber 0,5 Hz. Sind beide Bedingungen TRUE, arbeitet allein Taktgeber 1 Hz.

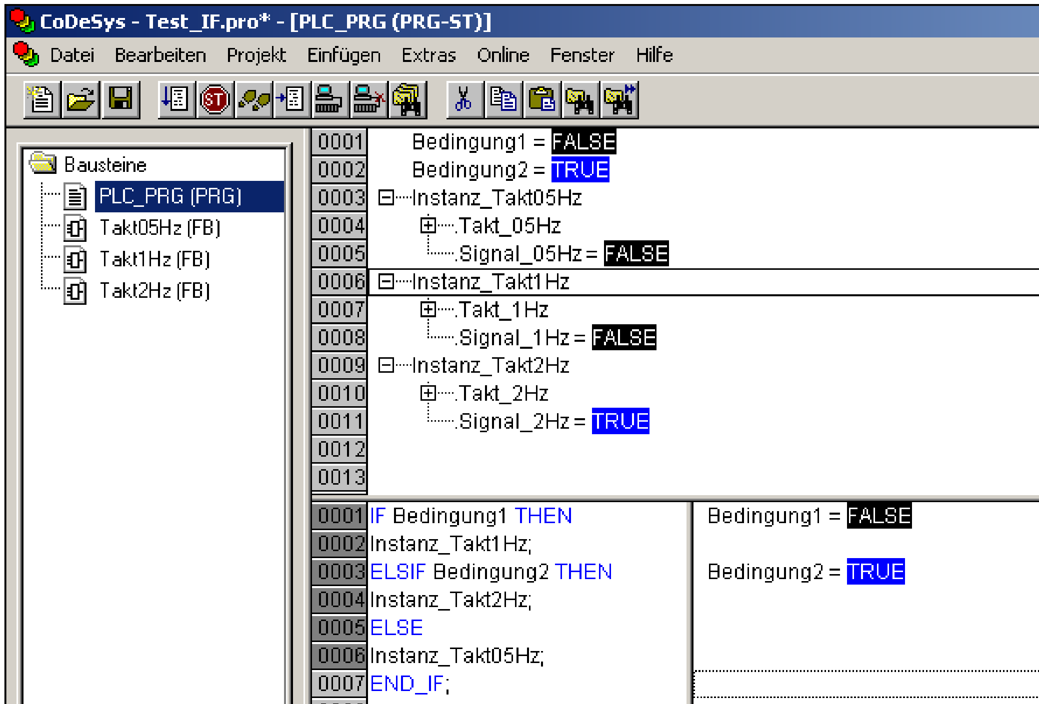


Bild 13-2: Online-Sicht der bedingten Abarbeitung von POE mit IF Anweisung für Bedingung2:=TRUE

Ein weiteres Beispiel für die Anwendung der IF Anweisung wird im Abschnitt 14 erläutert.

13.3 Aufruf von Funktion, Funktionsbaustein und Programm in ST

Nachfolgend wird der Aufruf von Funktion, Funktionsbaustein und Programm in ST denen in den Sprachen AWL und FUP gegenübergestellt.

- **Verwendung einer Funktion**

Vorgegeben sei eine Funktion „Meldung:BOOL" mit Rückgabewert vom Typ BOOL:

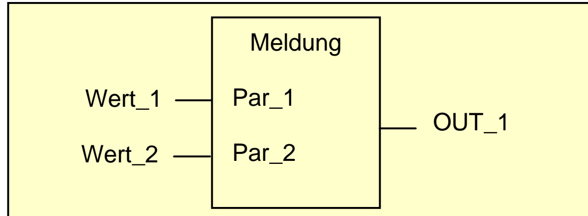
FUNCTION Meldung :BOOL VAR_INPUT Par_1:INT; Par_2:INT; END_VAR	Programm der Funktion in AWL: LD Par_1 ADD Par_2 GT 1200 ST Meldung	Programm der Funktion in ST: Meldung:=Par_1+Par_2>1200;

Bei gleichlautender Variablendeklaration lautet der Aufruf dieser Funktion in den unterschiedlichen Sprachen:

Deklaration für alle drei Fälle

```
VAR
Wert_1:INT;
Wert_2:INT;
OUT_1:BOOL;
END_VAR
```

Sprache FUP:



Sprache AWL:

```
LD Wert_1
Meldung Wert_2
ST OUT_1
```

Sprache ST:

```
OUT_1:=Meldung(Wert_1,Wert_2);
```

- **Aufruf (Instanzierung) eines Funktionsbausteins**

Vorgegeben sei nachfolgender Funktionsbaustein „Meldung“ :

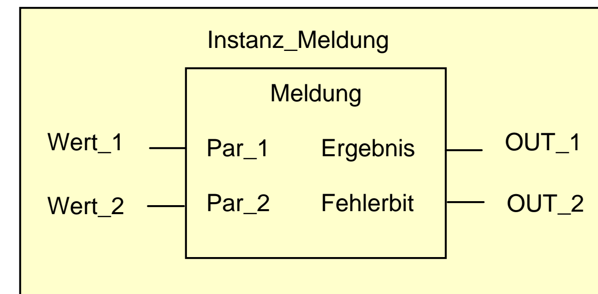
FUNCTION_BLOCK Meldung	Programm des FB in AWL:	Programm des FB in ST:
VAR_INPUT Par_1:INT; Par_2:INT; END_VAR VAR_OUTPUT Ergebnis:INT; Fehlerbit:BOOL; END_VAR	LD Par_1 ADD Par_2 ST Ergebnis GT 1200 ST Fehlerbit	Ergebnis:=Par_1+Par_2; Fehlerbit:=Ergebnis>1200 ;

Bei gleichlautender Variablendeklaration lautet der Aufruf (die Instanz) dieses Baustein in den unterschiedlichen Sprachen:

Deklaration für alle drei Fälle:

```
VAR
Instanz_Meldung:Meldung;
Wert_1:BOOL;
Wert_2:INT;
OUT_1:INT;
OUT_2:BOOL;
END_VAR
```

Sprache FUP:



Sprache AWL:

```
CAL Instanz_Meldung(Par_1:=Wert_1, Par_2:=Wert_2)
LD Instanz_Meldung.Ergebnis
ST OUT_1
LD INstanz_Meldung.Fehlerbit
ST OUT_2
```

Sprache ST:

```
Instanz_Meldung(Par_1:=Wert_1, Par_2:=Wert_2);
OUT_1:=Instanz_Meldung.Ergebnis;
OUT_2:= Instanz_Meldung.Fehlerbit;
```

- **Aufruf eines Programms**

Vorgegeben sei nachfolgendes Programm „Meldung“ :

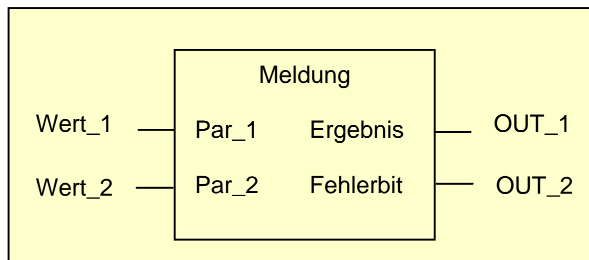
PROGRAM Meldung VAR_INPUT Par_1:INT; Par_2:INT; END_VAR VAR_OUTPUT Ergebnis:INT; Fehlerbit:BOOL; END_VAR	Programm in AWL: LD Par_1 ADD Par_2 ST Ergebnis GT 1200 ST Fehlerbit	Programm in ST: Ergebnis:=Par_1+Par_2; Fehlerbit:=Ergebnis>1200 ;

Bei gleichlautender Variablendeklaration lauten die Aufrufe dieses Programms in den unterschiedlichen Sprachen:

Deklaration für alle drei Fälle:

```
VAR
 Wert_1:BOOL;
Wert_2:INT;
OUT_1:INT;
OUT_2:BOOL;
END_VAR
```

Sprache FUP:



Sprache AWL:

```
CAL Meldung (Par_1:=Wert_1, Par_2:=Wert_2)
LD Meldung.Ergebnis
ST OUT_1
LD Meldung.Fehlerbit
ST OUT_2
```

Sprache ST:

```
Meldung(Par_1:=Wert_1, Par_2:=Wert_2);
OUT_1:=Meldung.Ergebnis;
OUT_2:=Meldung.Fehlerbit;
```

Der Aufruf eines Funktionsblocks oder eines Programms **ohne Parameterübergabe** kann geschrieben werden
Instanz_Name () oder aber nur *Instanz_Name*
bzw. Programm_Name() oder aber nur *Programm_Name*